Musica elettronica con il COMMODORE 64

Franco Fabbri

Beatrice d'Este
ARTI GRAFICHE RICORDI

Musica elettronica con il COMMODORE 64

Franco Fabbri

Beatrice d'Este
ARTI GRAFICHE RICORDI

IL COMPUTER *insegna*

SERIE EDUCATIONAL

Musica elettronica con il

COMMODORE 64

Franco Fabbri

Direttore responsabile: Giulio Sala
© 1984 by Arti Grafiche Ricordi - Milano
Registrato presso il Tribunale di Milano
N. 494 del 27 ottobre 1984
Prodotto dalla ENDA s.r.l.
Stampato in Italia presso:
Arti Grafiche Ricordi S.p.A.
Via B. Quaranta, 44 - Milano

TANTO PER COMINCIARE: POKE E PEEK

Se possedete il vostro Commodore 64 da qualche tempo, saprete già che molte delle notevoli risorse di questo computer (soprattutto per quanto riguarda la grafica e il suono) sono accessibili riferendosi a particolari locazioni di memoria. Il modo più semplice per farlo è di servirsi delle istruzioni Basic POKE e PEEK. Per fare solo un esempio tra i tanti, nel Basic standard del Commodore 64 non esiste un'istruzione specifica per stabilire il colore dello schermo o del bordo, ma, come insegna il manuale, occorre inserire in due locazioni di memoria (53280 e 53281) valori compresi tra 0 e 15, che rappresentano il colore scelto. Se non l'avete mai fatto, e avete il computer acceso, provate a battere sulla tastiera il comando POKE 53280,5 (poi RETURN). Cosa è successo? Ora battete POKE 53281,2. Cosa è successo? Provate a sostituire i numeri dopo la virgola con altri compresi tra 0 e 15. In qualche caso, i cambiamenti di colore dello schermo saranno tali che non riuscirete più a leggere quello che c'è scritto. Per tornare alla situazione iniziale, premete RUN/STOP e RESTORE, contemporaneamente. Se non avete il computer sottomano (i libri si leggono anche in treno, no?) prendete comunque nota: inserendo valori opportuni in certe locazioni di memoria si ottengono cambiamenti in quello che il computer ci fa vedere, o sentire; in termini tecnici, nell'output.

Ma che cos'è una locazione di memoria? E che cosa sono POKE e PEEK? I programmatori esperti, i divoratori di riviste specializzate, possono saltare al prossimo paragrafo, oppure fare, insieme agli altri, un piccolo ripasso.

Nel Commodore 64 ci sono 65536 locazioni di memoria, cioè (alla lettera) "luoghi" dove possono essere memorizzate delle informazioni. Queste ultime hanno dimensioni precise: se l'informazione è un numero decimale positivo, deve essere un numero intero (senza virgola), non più piccolo di zero (ovviamente, visto che è positivo) e non più grande di 255. Questa unità di informazione (che ci permette di rappresentare, ad esempio, numeri decimali interi positivi, compresi tra 0 e 255) si chiama *byte*. Dunque il Commodore 64 ha una memoria di 65536 bytes. A differenza che con i chilogrammi, "chilo" (abbreviato con K) in informatica sta per 1.024, non per 1000. Se fate 64 per 1.024 (PRINT 64 * 1024) otterrete 65536, che spiega cosa vuol dire quando si legge che il Commodore 64 ha 64K (chilobytes) di memoria. Poiché ogni locazione può essere raggiunta con altrettanta facilità delle altre (cioè la probabilità di accedere all'una o all'altra è identica), la memoria viene chiamata *ad accesso casuale*, o RAM (*Randorm Access Memory*).

Possiamo immaginarci la memoria RAM del Commodore 64 come un immenso casellario postale: ogni casella non può contenere più di

un tanto (solo un byte), ed è caratterizzata da un numero (da 0 a 65535: in totale 65536 caselle) che rappresenta il suo *indirizzo*. Possiamo anche immaginarci di avere un fattorino, che a nostra richiesta può andare alla casella numero tale e riportarci l'informazione che vi era contenuta, o inserirvi una nuova informazione cancellando quella vecchia.

In Basic queste due operazioni corrispondono rispettivamente alle istruzioni PEEK e POKE. *To peek*, in Inglese, vuol dire "sbirciare"; *to poke* vuol dire "spingere": con PEEK si può sbirciare cosa c'è in una locazione di memoria, con POKE si può spingere dentro una nuova informazione, cacciando fuori quella vecchia. In realtà il Basic fa qualcosa di più che - semplicemente - sbirciare o spingere: ad esempio, controlla che l'indirizzo sia giusto, e che l'informazione abbia le dimensioni giuste. Provate a battere POKE 70000,200 (*RETURN*), oppure POKE 50000,300 (*RETURN*): otterrete in risposta in entrambi i casi ?ILLEGAL QUANTITY ERROR. Cos'era sbagliato?

Verificate sul vostro manuale la sintassi corretta dei due comandi POKE e PEEK. Troverete che con POKE si deve separare con una virgola il primo numero (che rappresenta l'indirizzo di memoria) dal secondo (che rappresenta il valore decimale intero positivo che si vuole inserire); nel caso di PEEK, bisogna mettere tra parentesi il numero della locazione da sbirciare. PEEK da solo non ci dà nessuna informazione: per sapere il risultato bisogna "stamparlo" sul video. Ad esempio, PRINT PEEK (162).

Nelle 65536 locazioni di memoria del Commodore 64 non sono contenuti solo dati e programmi inseriti dall'utente, ma anche quei particolari programmi che garantiscono il funzionamento del computer: il *sistema operativo*, con l'*interprete* Basic. Senza addentrarci troppo, è ovvio che nel computer ci deve essere qualcosa che traduce una sequenza di caratteri battuti sulla tastiera (ad esempio, -P-R-I-N-T- -P-E-E-K-(-1-6-2-)-) in una serie di operazioni (controllare l'esattezza dell'istruzione, eseguirla oppure stampare un messaggio d'errore). Questo compito di traduzione spetta (chi l'avrebbe mai detto?) all'interprete. Ma avrete anche notato che il computer fa sempre qualcosa, anche quando non esegue istruzioni o programmi inseriti da voi: ad esempio, vi offre un'immagine sul video, con un cursore che lampeggia. Anche questo è il risultato di programmi che fanno parte del sistema operativo.

Questi programmi sono contenuti nella memoria sotto forma di numeri, inseriti in locazioni successive: alcuni di questi numeri rappresentano istruzioni, altri dati. Per cui capirete che non è la miglior cosa che si possa fare inserire (con l'istruzione POKE) valori a casaccio in locazioni a casaccio: dato che quando si "scrive" in una locazione si cancella quello che c'era prima, si rischia di bloccare il funzionamento del computer. Se avete lì il computer (e non avete programmi importanti in memoria), provate a battere POKE 788,0. Dove è andato a finire il cursore lampeggiante? Perché il computer non risponde più ai comandi (nemmeno a RUN/STOP RESTORE)? Ecco: abbiamo sostituito un dato che ser-

viva al computer per trovare il sottoprogramma che fa lampeggiare il cursore e legge la tastiera.

Perciò conviene sapere in quali parti della memoria si può scrivere (con la POKE), e cosa scrivere. I manuali offrono sempre una o più mappe della memoria, che segnalano quali gruppi di indirizzi (o zone di memoria) contengono che cosa. Questo manuale non sarà da meno.

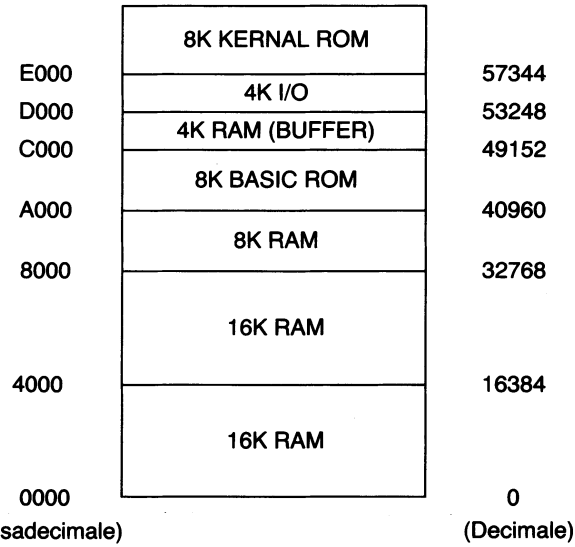

Fig. 1 - Mappa della memoria

L'istruzione PEEK è meno "pericolosa": sbirciare in una locazione non ne modifica il contenuto. Comunque, per quanto POKando numeri a caso possiate temporaneamente "uccidere" il vostro computer, la resurrezione è certa spegnendo e riaccendendo: il sistema operativo si trova in una memoria ROM (*Read Only Memory*: memoria a sola lettura) che all'accensione riversa le sue informazioni (quelle "giuste") nelle locazioni opportune. Voi potete scrivere in queste ultime, ma non - per definizione - nella ROM, che resta pronta a rimettere tutto a posto.

A questo punto, se volete, provate pure a leggere e a scrivere (sbirciare e spingere, PEEK e POKE) nelle locazioni di memoria che preferite. Nella maggior parte dei casi se scrivete in una locazione (con POKE) e poi andate a leggere (con PEEK) troverete il valore che avevate appena scritto. Ad esempio, POKE 49152,40 (*RETURN*), PRINT PEEK (49152) (*RETURN*). A questo comando, il computer ha risposto 40? Dovrebbe averlo fatto, se non avete sbagliato qualcosa. In altri casi potrete avere delle sorprese (ricordate sempre che se il computer si blocca basta spegnerlo e riaccenderlo). Tra i casi "bizzarri" (voi scrivete qualcosa, e quando andate a leggere nella stessa locazione trovate un altro valore) uno che ci interessa da vicino è quello delle 25 locazioni che vanno da 54272 a 54296. Qualunque cosa voi scriviate, quando poi andate a

leggere otterrete in risposta zero. Ad esempio, battete POKE 54272,40 (*RETURN*), e poi PRINT PEEK (54272) (*RETURN*). La risposta al secondo comando è 0, non 40. Perché?

Perché queste 25 locazioni sono "a sola scrittura" (*write only*): ci si può scrivere, ma non ci si può leggere. Qeste sono le locazioni di memoria che controllano il SID 6581, il formidabile generatore di suoni del Commodore 64.

IL COMMODORE 64 SUONA!

Facendo girare programmi già confezionati, o durante le vostre più o meno casuali POKate nella memoria del Commodore 64, vi sarete resi conto che il vostro computer può generare suoni e rumori. Niente di straordinario, in linea di principio: il suono è uno dei possibili output del Commodore 64, e non tra i più complessi. Avete pensato a quante operazioni possono essere necessarie solo per far apparire un rettangolo blu con una cornice azzurra, le scritte azzurre * * * * COMMODORE 64 BASIC V2 * * * *, eccetera, con un cursore che lampeggia? O per muovere il carrello e gli aghi di una stampante? Un microprocessore può far funzionare una lavatrice, *quindi* può anche suonare. Questo per dire che quelli che dicono che "far suonare" un computer è difficile (rispetto a fargli calcolare le tabelline), o sopravvalutano questo compito sonoro, oppure sottovalutano gli altri che il computer normalmente svolge. La questione è, piuttosto, che il Commodore 64 lascia molto al programmatore, non offrendogli soluzioni e comandi già pronti. Come non c'è un comando COLORB (COLORe del Background, cioè dello sfondo) che in altri computer stabilisce in modo chiaro e "amichevole" quale colore si deve assegnare allo sfondo, ma si deve fare POKE 53281,n (con n compreso fra 0 e 15), così non ci sono comandi SOUND, NOTE, FREQ, ENVELOPE, e simili, ma si devono fare le opportune POKE nelle 25 locazioni di memoria che controllano il generatore di suono. E perchè non proviamo a farle?

Spegnete (o immaginate di spegnere) il vostro computer. Se siete perplessi dell'utilità dell'operazione, pensate solo che serve ad azzerare le locazioni di memoria che controllano il generatore di suono. Poco più avanti troveremo un sistema più elegante.

Adesso battete, *nell'ordine in cui sono scritte*, le seguenti istruzioni:

POKE 54296,15 (*RETURN*)
POKE 54272,214 (*RETURN*)
POKE 54273,28 (*RETURN*)
POKE 54277,160 (*RETURN*)
POKE 54278,250 (*RETURN*)
POKE 54276,33 (*RETURN*)

Se non avete fatto sbagli, dovrebbe essere sucesso questo: 1) quando avete premuto RETURN per la prima volta (dopo POKE 54296,15) si è sentito un *pop* nell'audio del televisore; se avete solo un monitor senza audio leggete il manuale del computer (o questo stesso libro, piu avanti) per vedere come ci si collega a un amplificatore. 2) Dopo aver battuto l'ultimo RETURN, avete sentito un suono (per l'esattezza un La di 440 cicli per secondo, o Hertz), che è cominciato abbastanza gradualmente, per raggiungere un volume costante. Se avete resistito alla tentazione di spegnere il computer o di escludere l'audio (del televisore o dell'amplificatore), battete ora POKE 54276,32 (*RETURN*). Il suono dovrebbe scomparire gradualmente.

E' così difficile? Con poche istruzioni dirette, avete controllato il suono prodotto dal Commodore 64, non diversamente da come potreste fare per cambiare i colori dello sfondo e della cornice. Senza bisogno di memorizzare ancora una sola linea di programma, avete realizzato un diapason di altissima precisione, che potrebbe servire per accordare altri strumenti.

Delle locazioni di memoria che abbiamo usato, la prima (54296) controlla - in questo caso - il volume: il valore può variare tra 0 e 15, e come succede su molti apparecchi audio, portare bruscamente il volume al massimo o al minimo (ad esempio, accendendo o spegnendo un amplificatore col volume già alto) produce un *pop*. La seconda e la terza locazione (54272 e 54273) controllano l'altezza del suono prodotto. La quarta e la quinta (54277 e 54278) controllano la velocità con cui il suono sale al volume massimo o discende a zero. La sesta (54276) stabilisce se il suono deve iniziare o finire: abbiamo visto che inserendo il valore 33 il suono parte, inserendo 32 il suono termina (pensate a questa locazione come al tasto di un organo: premendolo il suono inizia, rilasciandolo il suono finisce).

Bene. Si può immaginare che a questo punto siate curiosi di vedere cosa succede cambiando i valori che abbiamo appena inserito. Siete liberi di farlo; senza necessariamente ribattere tutto, spostatevi sullo schermo con i cursori, correggete i numeri, battete RETURN. Se perdete il controllo, spegnete pure.

Sia che abbiate potuto sperimentare i cambiamenti o che ve li siate solo immaginati, avrete capito che ci troviamo davanti a uno strumento piuttosto complesso (non complicato: complesso...). Prima di vedere in dettaglio come è fatto, ci servono alcune nozioni generali sugli strumenti musicali elettronici. Ma prima ancora, per non finire il capitolo senza aver messo da parte qualcosa, ecco un programmino per richiamare il nostro diapason più facilmente.

```
PROGRAM:  1.DIAPASON

10 PRINT"[CLEAR]":PRINT"[WHITE]";
   CHR$(142):POKE 53280,2
   :POKE 53281,11
20 PRINT" PREMI <A> PER SUONARE,
    <B> PER FINIRE":PRINT
   :PRINT " (LA 440 HZ)"
30 FOR I=54272 TO 54296:POKE I,0:NEXT
40 POKE 54296,15
50 POKE 54272,214
60 POKE 54273,28
70 POKE 54277,160
80 POKE 54278,250
90 GET A$:IF A$="" GOTO 90
100 IF A$="A" THEN POKE 54276,33
   :GOTO 90
110 IF A$="B" THEN POKE 54276,32
   :GOTO 90
120 GOTO 90
```

Fate attenzione, per ora, alla linea 30: presenta un sistema per "pulire" le locazioni di memoria che controllano il generatore di suono (che d'ora in poi chiameremo *registri* del SID) più elegante ed efficiente che spegnere e riaccendere il computer. Nella maggior parte delle applicazioni inseriremo questa linea (o una simile) all'inizio del programma.

COS'E' UN SINTETIZZATORE?

A grandi linee un sintetizzatore è uno strumento che genera suoni elettronicamente, permettendo a chi lo usa un controllo più o meno sofisticato delle varie caratteristiche del suono. Insomma, non tutti gli strumenti che generano il suono elettronicamente sono sintetizzatori, anche se tra un sintetizzatore e l'altro ci possono essere differenze notevolissime. Il primo sintetizzatore che abbia mai portato questo nome, l'RCA Music Synthesizer, progettato a partire dal 1952, occupava una stanza intera; il Minimoog, costruito da Robert Moog negli anni Settanta, era grande come gli attuali strumenti elettronici a tastiera; il SID 6581, che è un vero e proprio sintetizzatore (simile nelle possibilità di controllo al Minimoog), è poco più grande di un francobollo. Eppure con il SID e il Commodore 64 che lo contiene si possono fare cose che i progettisti dell'RCA Music Synthesizer potevano solo sognare. Naturalmente, i confini dell'immaginazione oggi si sono spostati, e le capacità musicali del Commodore 64 sono davvero modeste se si confrontano con quelle di favolosi computer specializzati come il 4X, il Synclavier, il Fairlight CMI, o anche di strumenti commerciali di costo più accessibile, come i vari sintetizzatori polifonici esistenti. Ma proprio per il fatto di essere relativamente semplice, e tutto da programmare, il sintetizzatore del Commodore 64 ci permette di fare, a basso costo e a casa nostra, una grande quantità di esperimenti creativi che trent'anni fa mobilitavano interi laboratori universitari.

Quali sono le caratteristiche del suono che un sintetizzatore permette di controllare? Schematicamente, sono tre: l'intensità, l'altezza, l'andamento nel tempo.

Come sapete, il suono consiste in una vibrazione di un mezzo elastico, percepita dall'orecchio e tradotta in stimoli per il cervello. Una vibrazione altro non è che un'oscillazione rapidissima attorno a una posizione di riposo. Non tutte le oscillazioni sono percepibili come suoni: devono essere abbastanza ampie da poter sollecitare il nostro orecchio, ma non tanto da rompere il timpano; devono essere abbastanza rapide da non essere percepite come impulsi isolati, ma non tanto rapide da superare la capacità dell'orecchio di tradurle in stimoli nervosi; inoltre, perché l'orecchio e il cervello riescano a interpretare che cosa hanno sentito, è necessario che l'oscillazione duri almeno qualche frazione di secondo. Un sintetizzatore, quindi, deve essere in grado: 1) di produrre un'oscillazione che possa essere trasformata in una vibrazione meccanica; 2) di controllare l'ampiezza dell'oscillazione (che determina di quanto le particelle del mezzo elastico si sposteranno dalla loro posizione normale, di riposo); 3) di controllare la frequenza dell'oscillazione (quante volte l'oscillazione avviene in un secondo); 4) di controllare come l'oscillazione avviene nel tempo.

Avrete notato che non è stata ancora nominata una caratteristica del

suono che dovrebbe essere importantissima, parlando di sintetizzatori: il timbro. E' per ottenere timbri diversi da quelli degli strumenti tradizionali, tra l'altro, che si sono costruiti i sintetizzatori: perché non parlarne?

Certo, calma. In realtà lo abbiamo già fatto. Solo che per capirci qualcosa non serve accennare a suoni metallici, flautati, ovattati, percussivi, nasali e a tutti gli altri aggetivi con cui si descrive il timbro. Perché il timbro di un suono altro non è che l'andamento dell'oscillazione nel tempo. Visto che ne avevamo già parlato?

Questo piccolo manuale non è un trattato di elettroacustica, ma è necessario che sappiate almeno come l'oscillazione di una corrente elettrica può essere trasformata in un suono. Ciò avviene nell'altoparlante, che nella sua forma più semplice è costituito da un cono di cartone, da una bobina (un avvolgimento di filo conduttore) solidale con il cono (in genere è attaccata su una superficie cilindrica, che sta al posto della punta del cono), e da una calamita.

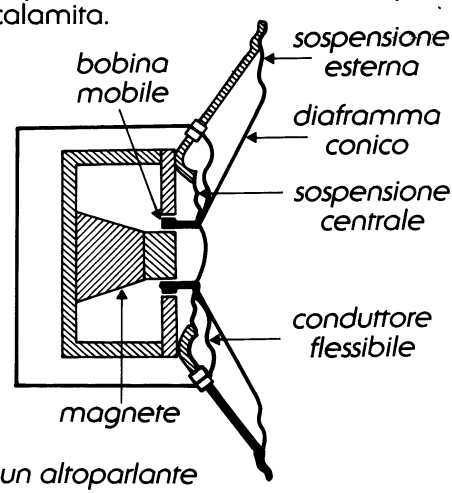

Fig. 2 Veduta in sezione di un altoparlante

"Oscillazione" riferito a una corrente, vuol dire che l'intensità e/o la direzione della corrente cambiano periodicamente; in altri termini, vuol dire che il potenziale agli estremi del conduttore cambia valore e/o segno (dove prima era negativo diventa positivo, e viceversa). La normale corrente alternata che alimenta le nostre lampadine, i nostri frullatori e i nostri computer è una corrente che oscilla, con una frequenza di 50 Hz: prima in uno dei buchi della spina c'è un potenziale positivo rispetto all'altro, poi negativo, e questo cambiamento avviene 50 volte ogni secondo (la differenza di potenziale in questo caso rimane sempre la stessa: 220 Volt).

Quando una corrente cambia intensità e/o direzione (quando una differenza di potenziale cambia valore e/o si inverte) avviene un cambiamento anche nel campo magnetico circostante: in particolare, se la corrente circola in una bobina, il cambiamento di direzione fa invertire la direzione di un campo magnetico perpendicolare al piano in cui si trova la bobina. La bobina percorsa dalla corrente, insomma, equivale

a una calamita: cambiando la direzione della corrente si scambiano il polo Nord e il polo Sud di questa calamita immaginaria.

Ora, se vicino alla bobina c'è invece una calamita vera, succede questo: se la corrente circola nella bobina in una certa direzione la calamita immaginaria (la bobina) e quella vera si attraggono, altrimenti si respingono. Chiaro?

Bene. Allora, visto che la bobina è appiccicata al cono di cartone, quest'ultimo si metterà a vibrare avanti e indietro (verso la calamita, o in direzione opposta) con la stessa frequenza con cui cambia la direzione della corrente (o la differenza di potenziale). Se la corrente oscilla con una frequenza di 50 Hz, anche il suono dell'altoparlante avrà la stessa frequenza.

Non vi venga in mente di attaccare un filo ai morsetti di un altoparlante, e di infilarlo in una presa di corrente. Gli altoparlanti, sono costruiti per sopportare tensioni massime di pochi Volt, quindi otterreste solo di fondere la bobina o sfasciare il cono. Ma il principio è questo. Ora, vi sarà chiaro che tutto quello che possiamo chiamare "musica elettronica" gira attorno a questo principio: generare una corrente oscillante che - amplificata se necessario - sia in grado di far vibrare un altoparlante producendo così un suono.

Un circuito che produce una corrente oscillante si chiama (incredibile a dirsi!) *oscillatore.* E' ovvio che il controllo sulle caratteristiche del suono di cui parlavamo all'inizio, tentando di definire cos'è un sintetizzatore, si traduce nel controllo della corrente oscillante generata dall'oscillatore. Un passo avanti decisivo nella realizzazione di veri sintetizzatori si ebbe nella seconda metà degli anni Sessanta, quando vennero introdotti gli *oscillatori a controllo in tensione* (*Voltage Controlled Oscillator*, VCO): sono circuiti nei quali la frequenza del segnale generato dipende dal valore di una tensione di controllo. Insieme ai VCF (filtri controllati in tensione), ai VCA (amplificatori controllati in tensione) e gli EG (*Envelope Generator*, generatore di inviluppo: vedremo cos'è) costituiscono i componenti di base, o *moduli*, di qualsiasi sintetizzatore.

Avrete notato che è comparso un termine nuovo, *segnale*. Ai nostri fini, per ora, non ha molta importanza se l'informazione sonora si trova sotto forma di vibrazione meccanica, oscillazione elettrica, sequenza di numeri: chiameremo segnale qualsiasi cosa porti quest'informazione.

Bene. Qual è la caratteristica che ha determinato il successo dei sintetizzatori costituiti da diversi moduli (inizialmente, VCO, VCA, VCF)? E' il fatto che il segnale che esce da un modulo può essere usato sia in quanto tale (per generare alla fine il suono), sia per controllare un altro modulo. E' lo stesso sintetizzatore, quindi, a fornire i complessi segnali che sono necessari per controllare il suono nel modo più sofisticato possibile, dove con "sofisticato" intenderemo accurato, ma anche rapido.

Ora passiamo a esaminare i moduli di un sintetizzatore ad uno ad uno: sarà l'occasione per chiarire definitivamente alcune questioni sulle caratteristiche del suono.

L'OSCILLATORE

Un sintetizzatore in genere contiene più di un oscillatore: un synth *monofonico* (che produce una sola nota per volta) può averne anche solo due, mentre uno *polifonico* (che può suonare più note, più voci contemporaneamente) avrà almeno tanti oscillatori quante voci.

Ogni oscillatore deve essere in grado di generare un segnale di frequenza compresa nell'estensione dello strumento: le frequenze udibili vanno da circa 16 a circa 20000 Hz (comunque, al di sopra dei 16000 Hz quello che si percepisce è un più o meno accentuato "chiarore" di suoni di frequenza inferiore, non un suono in quanto tale), ma le frequenze di base dei suoni che si usano abitualmente in musica non superano le poche migliaia di Hz. Oltre alla frequenza (che deve essere controllabile con la massima precisione e rapidità attraverso un segnale esterno) è importante il modo con cui avviene l'oscillazione nel tempo, che determina la *forma d'onda*.

Nell'*onda triangolare* (*triangle wave*) il segnale passa a velocità costante dal valore massimo a quello minimo, poi inverte istantaneamente la direzione e si riporta, alla stessa velocità, di nuovo al valore massimo. Quello che abbiamo appena descritto è un *ciclo* dell'onda triangolare.

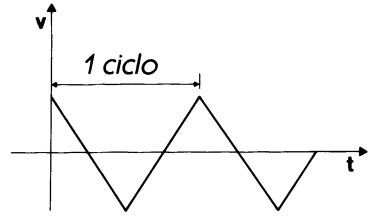

Fig. 3
Onda triangolare

Nell'*onda a dente di sega* (*sawtooth wave*) il segnale passa a velocità costante dal valore massimo a quello minimo, ma quando arriva lì istantaneamente si riporta al valore massimo. E' un'onda a dente di sega anche quella simmetrica all'onda che abbiamo appena visto, cioè quella in cui il segnale cresce a velocità costante dal minimo al massimo, e poi precipita istantaneamente al minimo.

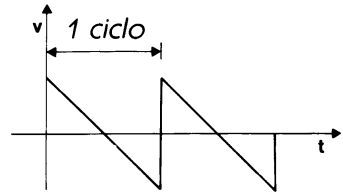

Fig. 4
Onda a dente di sega

Nell'*onda quadra* (*square wave*) il segnale si mantiene costante per un po' al livello massimo, poi precipita al minimo, dove resta per un altro po', finchè ritorna altrettanto rapidamente al massimo. Se il periodo

di tempo durante il quale il segnale si trattiene al livello massimo è diverso da quello in cui resta al minimo, l'onda quadra diventa un'*onda a impulsi* (*pulse wave*).

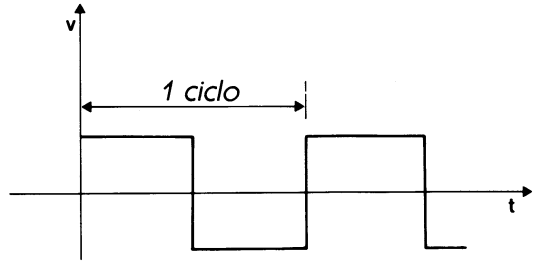

Fig. 5
Onda quadra

Infine, nell'*onda sinusoidale* (*sine wave*) il segnale va dal massimo al minimo e viceversa con un moto armonico. Per farvene un'idea, pensate al movimento di un pendolo da un estremo all'altro, e viceversa: il pendolo prima si muove lentamente, poi accelera, raggiunge la velocità massima quando passa per la posizione di equilibrio, poi rallenta sempre più, si ferma per un istante infinitesimo, riparte nell'altra direzione.

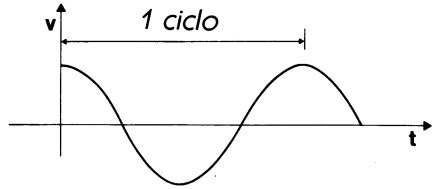

Fig. 6
Onda sinusoidale

Avevamo visto che il modo con cui avviene l'oscillazione nel tempo determina il timbro: questo vuol dire che il timbro è determinato, tra l'altro, dalla forma dell'onda. Le onde che abbiamo appena descritto hanno timbri diversi: per rendervene conto, battete (o prendete dalla cassetta) e fate girare questo piccolo programma: come vedete, è una semplice estensione del primo che avevamo fatto.

```
PROGRAM:  2.FORMA D'ONDA

 10 PRINT"[CLEAR]":PRINT"[WHITE]";
    CHR$(142):POKE 53280,2
    :POKE 53281,11
 20 PRINT"PREMI <A> PER SUONARE, E/O:"
 30 PRINT:PRINT "<T> PER L'ONDA TRIANG
    OLARE"
 40 PRINT "<S> PER L'ONDA A DENTE DI S
    EGA"
 50 PRINT "<Q> PER L'ONDA QUADRA
 60 PRINT:PRINT "<B> PER FINIRE"
 70 FOR I=54272 TO 54296:POKE I,0:NEXT
```

```
 80 POKE 54296,15
 90 POKE 54272,214
100 POKE 54273,28
110 POKE 54274,0
120 POKE 54275,8
130 POKE 54277,160
140 POKE 54278,250
150 W=17:A$="T":GOSUB 230
160 GET A$:IF A$="" GOTO 160
170 IF A$="A" THEN POKE 54276,W
   :GOTO 160
180 IF A$="B" THEN POKE 54276,
   W AND 254:GOTO 160
190 IF A$="T" THEN W=17:POKE 54276,W
   :GOSUB 230:GOTO 160
200 IF A$="S" THEN W=33:POKE 54276,W
   :GOSUB 230:GOTO 160
210 IF A$="Q" THEN W=65:POKE 54276,W
   :GOSUB 230:GOTO 160
220 GOTO 160
230 PRINT"[HOME]";"[DOWN10]";
   "          ";"[HOME]";"[DOWN10]";
   "ADESSO:";A$
240 RETURN
```

Il programma vi può presentare solo tre delle quattro forme d'onda (se date un occhiata alle linee 190,200 e 210 vedrete che corrispondono a valori diversi di una variabile): gli oscillatori del Commodore 64 non possono generare segnali sinusoidali puri. Vi renderete conto, comunque, della differenza notevole fra i timbri, al punto che, giocando sui tasti "T", "S" e "Q", potreste ottenere sequenze di suoni che abbiano un senso musicale, per quanto basate su una sola nota.

Noterete anche che l'onda triangolare corrisponde a un suono più cupo, apparentemente di volume più basso, quasi più grave (di altezza minore), rispetto alle altre due forme d'onda. Questo deriva dal fatto che l'onda triangolare è più povera di *armoniche*. Cosa vuol dire?

Senza trasformare questo libro in un manuale di acustica, ecco quello che possiamo dire: è stato dimostrato da Fourier (un matematico francese del secolo scorso) che un'onda di qualsiasi forma può essere scomposta nella somma di infinite onde sinusoidali, che si chiamano armoniche. Le armoniche hanno frequenze multiple intere della frequenza dell'onda in questione: una volta, due volte, tre volte, e così via, all'infinito. Per ottenere una determinata forma d'onda, occorre che ognuna delle armoniche abbia, rispetto alle altre, una certa ampiezza. Detto come se si trattasse di una ricetta, ci vorrà un po' della prima (che si chiama *fondamentale*), un po' della seconda, un po' della terza eccetera, dove questo "po'" varia nelle diverse ricette per ottenere forme d'onda differenti. Può anche essere zero, vale a dire che in pratica non sono sempre necessarie infinite armoniche, ma ne bastano alcune; soprattutto, il contributo di certe armoniche può essere così ridotto che è lecito considerarlo nullo.

Così avviene, in particolare, per l'onda triangolare, che ha solo le armoniche dispari (la prima o fondamentale, la terza, la quinta, eccetera) e con un'intensità talmente decrescente, man mano che si va avanti nel numero dell'armonica, che si può quasi dire che l'onda triangolare coincida con la sua fondamentale sinusoidale. Mancando le armoniche di frequenza più alta, l'onda triangolare corrisponde a un suono più cupo.

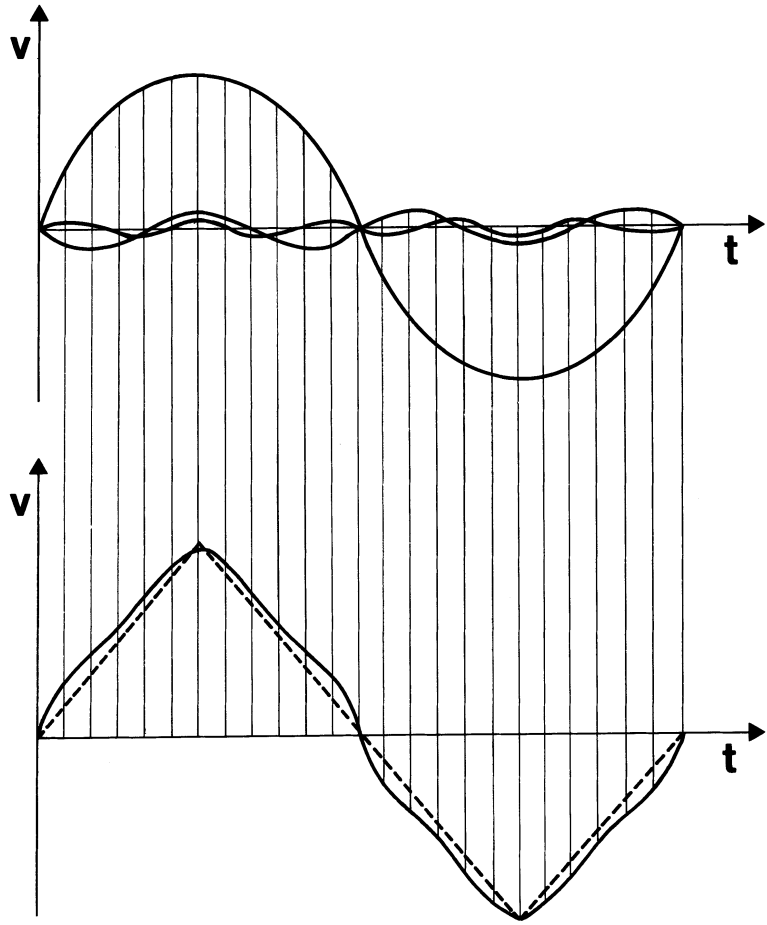

Fig. 7 Le prime tre armoniche di un'onda triangolare

Tra le numerose tecniche di sintesi, da quello che abbiamo appena visto ne possiamo dedurre almeno due: la *sintesi addittiva*, che consiste in pratica nel ricreare la forma d'onda desiderata a partire dalle singole armoniche, e la *sintesi sottrattiva*, che si fa partendo da un suono ricco di armoniche e togliendo quelle che non servono. La maggior parte dei sintetizzatori commerciali è basata su quest'ultimo tipo di sintesi, e così anche il SID. Ma come si fa a togliere le armoniche che non servono?

IL FILTRO

Un filtro è un sistema che lascia passare segnali di una certa frequenza e blocca quelli di frequenza diversa. Dato che ora sapete (quasi) tutto sulle armoniche, non vi sarà difficile immaginare che in realtà un filtro agisce sulle componenti frequenziali di un segnale complesso, cambiando di conseguenza la sua forma d'onda.

Ci sono vari tipi di filtri: *passa basso* (*low pass*) se possono passare le frequenze basse e non quelle alte; *passa alto* (*high pass*), viceversa; *passa banda* (*band pass*) se può passare un insieme, o banda, di frequenze, e le altre no; *a reiezione di banda* (*band reject*) se passano tutte le frequenze, ad eccezione di quelle comprese in una certa banda. La frequenza alla quale inizia il taglio, oppure la frequenza centrale della banda che passa o che viene respinta, deve essere regolabile (in un VCF è controllata dalla tensione). Nei sintetizzatori commerciali (e nel SID) è presente anche un controllo della *risonanza*, cioè un'esaltazione delle frequenze vicine a quella d'intervento.

Oltre che dalla modalità e dalla frequenza di intervento, un filtro è caratterizzato dalla *pendenza*, cioè dalla rapidità (lungo l'asse delle frequenze) con cui si passa dalla trasmissione del segnale al taglio. La pendenza si esprime in dB/ottava. Se queste unità di misura vi sconvolgono, eccone una spiegazione rudimentale: il *decibel* (abbreviato con dB) è uguale a dieci volte il logaritmo del rapporto tra le potenze di due segnali, o venti volte il logaritmo del rapporto tra le loro intensità. Insomma, più due segnali differiscono tra loro in potenza o in intensità, e più la misura in decibel di questa differenza aumenta in valore assoluto. L'ottava è l'intervallo tra due suoni (o segnali) che hanno uno frequenza doppia dell'altro. Se un filtro ha una pendenza di 12 dB/ottava vuol dire che salendo di un ottava in corrispondenza della frequenza di taglio l'intensità del segnale cambia di 12 dB. Non è difficile vero?

Per avere un idea dell'influenza che un filtro può esercitare su un segnale dato, fate girare il nostro terzo programma, del quale ecco il listato:

```
PROGRAM:  3.FILTRO

 10 PRINT"[CLEAR]":PRINT"[WHITE]";
    CHR$(142):POKE 53280,2
    :POKE 53281,11
 20 PRINT" PREMI <A> PER SUONARE,
     <B> PER FINIRE":PRINT
    :PRINT " (FILTRO)"
 30 FOR I=54272 TO 54296:POKE I,0:NEXT
 40 POKE 54296,31
 50 POKE 54272,53
 60 POKE 54273,7
 70 POKE 54277,160
 80 POKE 54278,250
```

```
90 POKE 54294,255:POKE 54295,241
100 GET A$:IF A$="" GOTO 100
110 IF A$="A" THEN POKE 54276,33
   :GOSUB 140:GOTO 100
120 IF A$="B" THEN POKE 54276,32
   :GOTO 100
130 GOTO 100
140 FOR K=255 TO 0 STEP -1
150 POKE 54294,K
160 NEXT K
170 RETURN
```

Il suono iniziale è un La di 110 Hz, con una forma d'onda a dente di sega. Man mano che procede viene "spazzolato" da un filtro passa basso, che inizia ad intervenire sulle altre frequenze e poi scende progressivamente. Il suono finale che sentite assomiglia molto di più a quello di un'onda triangolare che al dente di sega originale. Lo scorrimento della frequenza di intervento del filtro verso il basso è controllato dalla breve subroutine delle linee 140-170: il tempo impiegato per passare dal timbro iniziale a quello finale corrisponde esattamente a quello necessario per svolgere le operazioni comprese nella subroutine.

Nel suo insieme, il suono che viene prodotto da questo programma potrebbe essere definito come un timbro, un timbro "miagolato". Insomma, solo un musicista elettronico incallito lo definirebbe come "l'evoluzione di un onda a dente di sega verso una quasi-sinusoide, per l'intervento di un filtro passa basso a frequenza di intervento discendente, con risonanza". Ancora una volta, vedete che quella qualità percettiva del suono che chiamiamo timbro dipende dall'evoluzione di quache suo parametro nel tempo. In ogni sintetizzatore c'è un modulo specializzato per controllare aspetti essenziali di questa evoluzione: ne parliamo nel prossimo paragrafo.

IL GENERATORE D'INVILUPPO (E L'AMPLIFICATORE)

Prendiamo il grafico di un'oscillazione che varia in ampiezza. Se congiungiamo tra loro tutti i massimi e tutti i minimi, otteniamo due curve simmetriche che indicano come cambia l'ampiezza dell'oscillazione al passare del tempo. E' come se avessimo avvilupato il grafico con una carta da imballo: questo è, grosso modo, il concetto di inviluppo di ampiezza (dato che le due curve sono simmetriche, una sola basta a descrivere l'andamento dell'ampiezza nel tempo). Più in generale, *inviluppo* è una curva che indica il variare di un parametro nel tempo: può essere l'ampiezza, ma anche la frequenza del suono, o il punto di intervento di un filtro.

Generare un inviluppo significa produrre un segnale di controllo per vari parametri (quelli che abbiamo appena visto), con un preciso decorso temporale, a sua volta controllabile: questo è ciò a cui serve il *generatore di inviluppo* (EG, *Envelope Generator*).

Gli inviluppi dei suoni reali sono molto complessi, e altrettanto complesso deve essere un generatore di inviluppo capace di riprodurli. In genere, per ragioni tecniche ed economiche, i generatori incorporati sui sintetizzatori commerciali creano inviluppi abbastanza schematici. L'inviluppo-tipo di questi strumenti (anche del SID) si può descrivere così:

1) una fase iniziale, nella quale il segnale di controllo passa da zero al suo valore massimo: *attacco*;

2) una seconda fase, nella quale il segnale di controllo decade dal massimo ad un secondo valore: *decadimento*;

3) una terza fase, in cui il segnale si mantiene al livello raggiunto al termine del decadimento: *sostegno*;

4) una quarta fase, in cui il segnale ricade definitivamente a zero: *rilascio*.

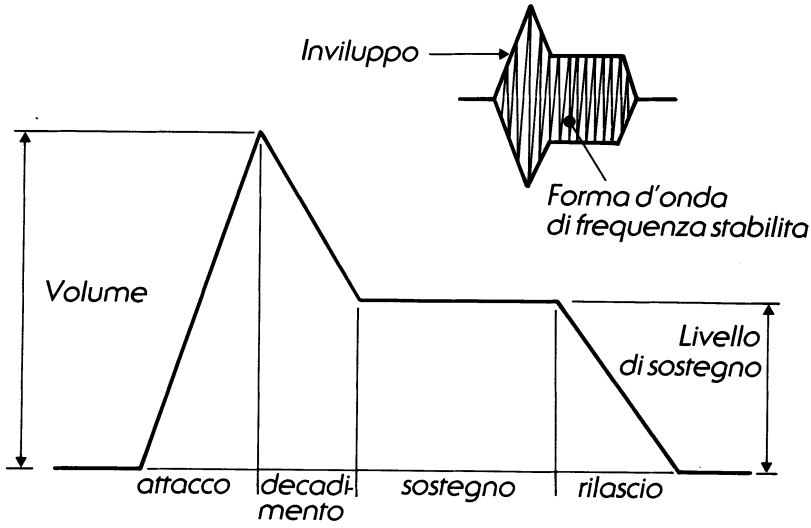

Fig. 8 Un inviluppo del tipo ADSR

Dalle iniziali delle fasi in cui si divide, questo tipo di inviluppo viene chiamato ADSR; a volte, erroneamente, ADSR è usato come sinonimo di EG.

L'inviluppo di ampiezza ha un'importanza estrema nel definire il carattere timbrico di un suono: i suoni percussivi hanno tempi di attacco molto rapidi, seguiti da un decadimento più o meno lento, e in genere senza sostegno; strumenti come gli archi o i fiati hanno attacchi più lenti, decadimento poco avvertibile, fasi di sostegno lunghe a piacere ("entro certi limiti!", dirà il trombonista), rilascio relativamente rapido; un organo elettrico ha un attacco praticamente immediato, seguito subito dalla fase di sostegno: poi, appena si lascia il tasto, il suono cessa.

Il generatore di inviluppo viene a sua volta controllato da un segnale, che conviene vi ricordiate perchè vi farà certamente disperare: si chiama *gate* (che vuol dire "soglia"). Normalmente, quando questo segnale di gate viene dato il generatore d'inviluppo inizia la fase di attacco, proseguendo per conto suo e stabilizzandosi nella fase di sostegno; quando il gate viene tolto, il generatore fa partire la fase di rilascio. *Prendete attentamente nota* del fatto che ciò che fa partire o terminare il segnale del generatore di inviluppo non è la presenza o l'assenza in sé del segnale di gate, ma il fatto che questo cambi; insomma, se state premendo un tasto di un organo con un dito e aggiungete un altro dito, non succede proprio niente: per fare due note staccate dovete prima togliere il dito e poi ripremere il tasto.

Normalmente, dovendo controllare l'ampiezza, il generatore di inviluppo è collegato a un amplificatore (nel caso di un sintetizzatore a controllo in tensione, a un VCA); ma per ottenere effetti timbrici interessanti si deve poter collegare l'EG anche ai filtri o agli oscillatori. Questo è un caso particolare di quelle possibilità di controllo incrociato, tra un modulo del sintetizzatore e l'altro, alle quali si era accennato in precedenza.

MODULATORI

Come abbiamo già visto, il segnale in uscita da un modulo del sintetizzatore può essere usato per controllarne un altro. Supponendo di avere a che fare con moduli a controllo in tensione, possiamo pensare di collegare un VCO (chiamiamolo VCO1) a un VCA, in modo che la quantità di amplificazione (il *guadagno*) del VCA sia regolata dal segnale oscillante del VCO1. Se nel VCA sta passando il segnale di un altro VCO (VCO2), l'ampiezza del segnale in uscita varierà con la stessa frequenza del VCO1. Questa procedura si chiama *modulazione di ampiezza*: se la frequenza del VCO *modulante* (il VCO1) è abbastanza bassa, l'effetto che ne deriva è il *tremolo*. Molti sintetizzatori (non il SID) contengono oscillatori specializzati a bassa frequenza (LFO, *Low Frequency Oscillator*) per questi effetti di modulazione.

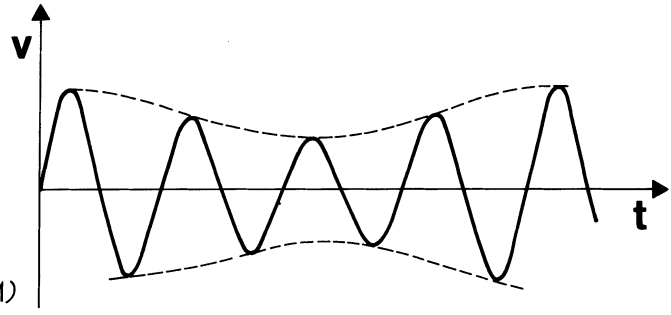

Fig. 9 Tremolo (AM)

Un VCO (o un LFO) può controllare anche un altro VCO: in questo caso abbiamo la *modulazione di frequenza*. L'onda che viene modulata si chiama *portante*, l'altra si chiama sempre modulante. A seconda dei valori della frequenza portante e di quella modulante e del grado di modulazione si possono ottenere risultati molto diversi: dal *vibrato* (che è una modesta modulazione dell'altezza di un suono, da parte di una bassa frequenza) a timbri molto complessi, che possono servire per la sintesi di segnali totalmente diversi da quelli di partenza. La modulazione di frequenza, per esempio, è la tecnica di sintesi impiegata dai sintetizzatori Yamaha della serie DX.

Infine, due segnali possono essere sottoposti a un processo di moltiplicazione, che dà in uscita un segnale complesso, formato dalla somma

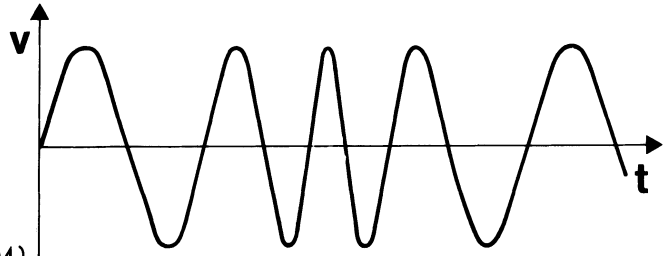

Fig. 10 Vibrato (FM)

e dalla differenza delle frequenze in entrata. Questa è la *modulazione ad anello*. Poiché difficilmente la somma e la differenza di due frequenze stanno in relazioni armoniche (cioè sono multiple intere di una stessa fondamentale, come le armoniche di uno stesso suono), i suoni modulati ad anello hanno un carattere inarmonico, "stonato", simile a quello degli oggetti di metallo percossi (lastre, tubi, campane).

I due programmi che seguono realizzano nel modo più semplice possibile una modulazione di ampiezza e una modulazione di frequenza, producendo rispettivamente un effetto di tremolo e uno di vibrato. Come vedete la struttura del programma è sempre la stessa: è una scelta precisa, per permettervi di cogliere le differenze. Naturalmente, a questo punto, alcune istruzioni o alcuni valori vi lasceranno perplessi: ma a programmare davvero il SID cominceremo più avanti.

```
PROGRAM:  5.VIBRATO

 10 PRINT"[CLEAR]":PRINT"[WHITE]";
    CHR$(142):POKE 53280,2
    :POKE 53281,11
 20 PRINT"PREMI <A> PER SUONARE"
 30 PRINT:PRINT "(QUESTO E' IL VIBRATO
    )"
 40 FOR I=54272 TO 54296:POKE I,0:NEXT
 50 POKE 54296,143
```

```
60 POKE 54272,214:POKE 54286,50
70 POKE 54273,28
80 POKE 54277,160
90 POKE 54278,250
100 GET A$:IF A$="" GOTO 100
110 IF A$="A" THEN POKE 54276,33
   :POKE 54290,17:GOSUB 140
120 POKE 54276,32:POKE 54290,16
   :GOTO 100
130 GOTO 100
140 FOR I=0 TO 500:POKE 54272,
   PEEK(54299):NEXT
150 RETURN
```

PROGRAM: 4.TREMOLO

```
 10 PRINT"[CLEAR]":PRINT"[WHITE]";
    CHR$(142):POKE 53280,2
    :POKE 53281,11
 20 PRINT"PREMI <A> PER SUONARE"
 30 PRINT:PRINT "(QUESTO E' IL TREMOLO
    )"
 40 FOR I=54272 TO 54296:POKE I,0:NEXT
 50 POKE 54296,15
 60 POKE 54272,214:POKE 54286,50
 70 POKE 54273,28
 80 POKE 54277,160
 90 POKE 54278,250
 100 GET A$:IF A$="" GOTO 100
 110 IF A$="A" THEN POKE 54276,33
    :POKE 54290,17:GOSUB 140
 120 POKE 54276,32:POKE 54290,16
    :GOTO 100
 130 GOTO 100
 140 FOR I=0 TO 500:POKE 54296,
    (PEEK(54299)/32+8) OR 128:NEXT
 150 RETURN
```

Qualcosa, comunque, si può anticipare: ammesso che un oscillatore debba in qualche modo "leggere" il segnale che lo controlla, e ammesso che nel Commodore 64 i segnali di controllo siano numeri, in che punto di ciascuno dei due programmi si va a "sbirciare" qualche cosa?

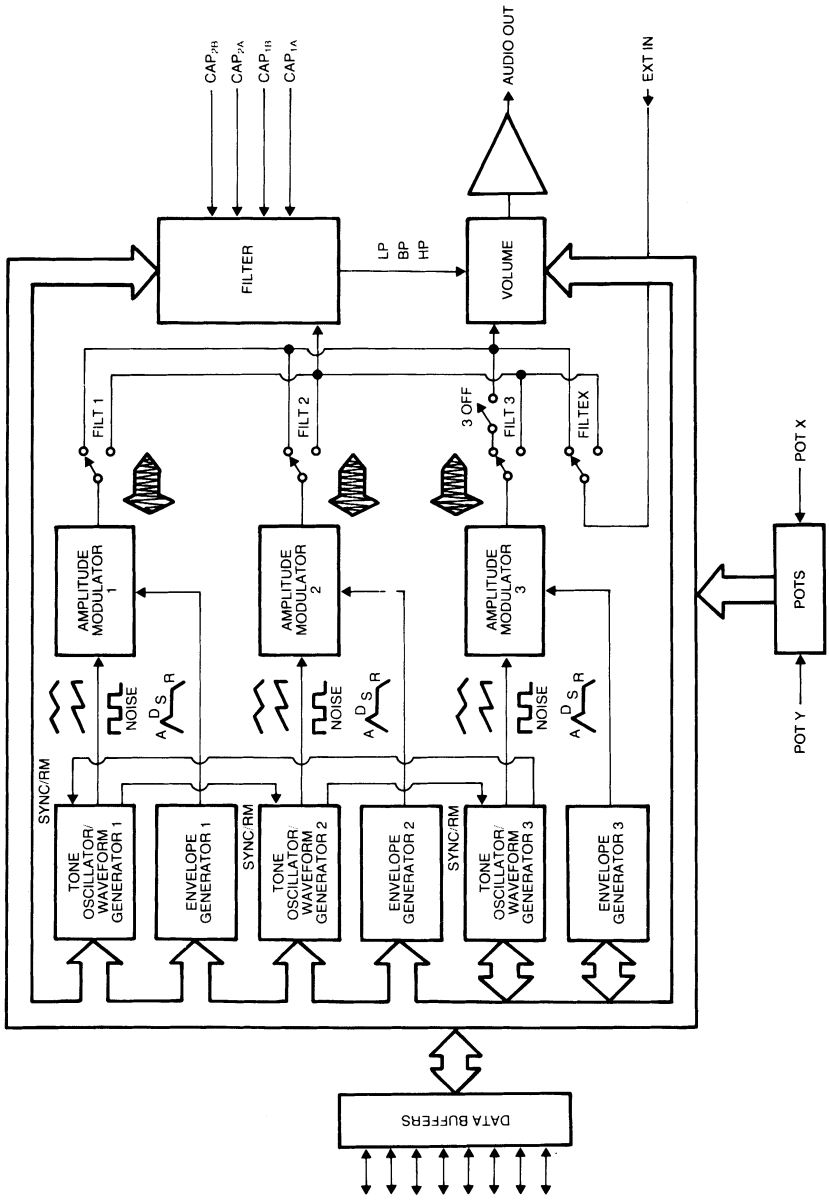

Fig. 11
Schema a blocchi del SID 6581

IL SID 6581

Il SID 6581 (*Sound Interface Device*) è un circuito integrato specializzato nella generazione e nel trattamento di segnali della gamma acustica, progettato da Bob Yannes nei laboratori della MOS Technology, la stessa ditta che fornisce l'unità centrale (6510a) e gli altri chip principali del Commodore 64.

E' dotato di tre oscillatori, ciascuno dei quali può generare quattro forme d'onda (più le combinazioni di alcune di queste): triangolare, a dente di sega, a impulsi (compreso il caso particolare della quadra), rumore. La frequenza fondamentale massima è di 3906 Hz, quella minima è virtualmente di 0 Hz (al di sotto dei 16 Hz il suono viene percepito come una serie di impulsi).

All'uscita di ogni oscillatore si trova un modulatore di ampiezza (con la funzione che in un sintetizzatore a controllo in tensione è svolta da un VCA), controllato da un generatore di inviluppo. La gamma dinamica del segnale che esce da ogni modulatore di ampiezza (cioè il rapporto tra il segnale più forte e il rumore di fondo, espresso in forma logaritmica) è di 48 dB. Il SID, quindi, è uno strumento abbastanza rumoroso: la dinamica che si richiede agli strumenti elettronici in genere supera i 70 dB. Del resto con il sistema di generazione di suono del SID (e al suo costo) non è possibile ottenere di più.

Ognuno dei tre generatori di inviluppo può controllare tempi di attacco tra 2 millisecondi e 8 secondi, e tempi di decadimento e rilascio tra 6 millisecondi e 24 secondi; il livello di sostegno varia, in sedici gradini, tra lo zero e il massimo.

I segnali provenienti dai tre oscillatori, più un eventuale segnale proveniente dall'esterno, possono essere inviati a un filtro, che prevede tutte le modalità di funzionamento (passa basso, passa alto, passa banda, a reiezione di banda), una frequenza centrale di intervento regolabile tra 0 e 3906 Hz, una pendenza di 12 dB/ottava (quindi un campo di intervento che si estende fino a oltre 12000 Hz), risonanza regolabile in sedici valori distinti.

Prima dell'uscita si trova un controllo del volume generale, regolabile in sedici valori: l'uscita è singola (mono).

Un'occhiata allo schema a blocchi fornito dalla Commodore sulla *Guida di riferimento* ci può dare un'idea dei percorsi del segnale all'interno del SID.

Come si vede, nello schema si distinguono due tipi di percorso: quello rappresentato da linee sottili, che è il percorso del segnale vero e proprio, e quello rappresentato da strisce più larghe, a forma di freccia, che è il percorso dei dati, i dati numerici attraverso i quali si controlla il SID. Nei prossimi due paragrafi vedremo questi aspetti separatamente.

I PERCORSI DEL SEGNALE

Il segnale proveniente da ciascuno degli oscillatori (TONE OSCILLATOR/ WAVEFORM GENERATOR) passa al modulatore di ampiezza (AMPLITUDE MODULATOR), che è controllato da un generatore di inviluppo (ENVELOPE GENERATOR). Come si vede, da ogni oscillatore esce anche un altro segnale (SYNC/RM), che va verso uno solo degli oscillatori: dall'oscillatore 1 all'oscillatore 2, dall'oscillatore 2 all'oscillatore 3, dall'oscillatore 3 all'oscillatore 1. Questi sono segnali di controllo, che permettono la modulazione ad anello o la sincronizzazione tra i segnali dei due oscillatori coinvolti.

Dopo i modulatori di ampiezza il segnale arriva a tre deviatori (indicati come FILT 1, 2 e 3), che lo smistano direttamente al regolatore di volume (VOLUME) oppure al filtro (FILTER); c'è anche un deviatore FILTEX, che permette eventualmente di inviare al filtro un segnale proveniente dall'esterno (EXT IN). Lungo il percorso del segnale dell'oscillatore 3 c'è anche un deviatore indicato come 3 OFF, che serve a impedire che questo segnale compaia all'uscita quando è usato invece per controllare altri moduli. Come abbiamo visto, il SID non ha un LFO, ma si può usare in questa funzione l'oscillatore 3 (i due programmini per ottenere il tremolo e il vibrato facevano proprio questo). Inoltre, i generatori di inviluppo del SID sono collegati stabilmente ai modulatori di ampiezza: questo non vuol dire che non si possano creare inviluppi di frequenze o di forme d'onda, controllando con un EG gli oscillatori o il filtro. Ma questo controllo avviene attraverso i dati, cioè lungo il percorso indicato dalla striscia; quindi ne parleremo tra poco.

Il segnale in uscita dal blocco VOLUME viene amplificato (il triangolo è il simbolo per uno stadio di amplificazione), e viene poi reso disponibile: 1) come componente audio del segnale per un televisore (accessibile dal jack RCA: è il collegamento più comune); 2) su due piedini di una presa DIN che si trova sul retro, tra il jack del televisore e la presa seriale. Nel secondo caso, collegando la massa (calza) di un normale cavetto schermato al piedino centrale (numero 2), e la linea al primo piedino a sinistra (numero 3), si ottiene un collegamento compatibile con le linee AUX di un normale amplificatore ad alta fedeltà.

Pin	Segnale
1	LUMINANZA
2	MASSA
3	AUDIO OUT
4	VIDEO OUT
5	AUDIO IN

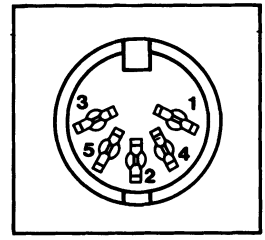

Fig. 12
Schema della presa audio-video.

Il segnale che si ottiene da questa seconda uscita è molto più pulito rispetto a quello del televisore, e ha il vantaggio di poter essere sottoposto a trattamenti ulteriori che ne migliorino la qualità: equalizzazione, espansione o *gating* (riduzione del rumore di fondo), riverberazione, eco. Il segnale che il SID accetta in ingresso attraverso EXT IN è di livello piuttosto elevato, paragonabile a quello delle uscite linea di un amplificatore o di un mixer: quindi non è possibile collegare direttamente al SID un microfono, una chitarra elettrica o altri dispositivi che generino segnali a basso livello; è necessario, prima, sottoporre questi segnali ad amplificazione. Un mixer, anche di pretese modeste, può essere un complemento utilissimo per trasformare il Commodore 64 in un minilaboratorio di musica elettronica.

I PERCORSI DEI DATI

I dati che controllano il SID trovano posto in venticinque registri interni al chip: ciascuno di questi corrisponde a una locazione di memoria nella RAM, a partire dal numero 54272. Come abbiamo visto si tratta di locazioni a sola scrittura, e questo è visibile sullo schema a blocchi dal fatto che la maggior parte delle frecce che indicano il percorso dei dati vanno in una sola direzione: dall'esterno all'interno.

Ci sono, però, delle eccezioni: vicino all'oscillatore 3 e al suo generatore di inviluppo c'è una doppia freccia, a significare che questi moduli non solo accettano dati ma sono anche in grado di fornirne all'esterno. Questo, tuttavia, non avviene attraverso gli stessi registri, ma per mezzo di due registri addizionali, il cui contenuto può essere letto nelle due locazioni a sola lettura 54299 e 54300. Queste locazioni comunicano rispettivamente lo stato dell'oscillatore 3 (cioè il punto a cui si trova l'oscillazione nel momento in cui avviene la lettura) e lo stato dell'inviluppo (cioè il punto dell'inviluppo al quale si è arrivati al momento della lettura). Ne riparleremo più avanti; comunque, se adesso andate a riprendere i programmi del tremolo e del vibrato, vedrete che l'unica PEEK contenuta andava proprio a "sbirciare" nella locazione 54299, travasandone il contenuto (tale e quale nel caso del vibrato, un po' rielaborato in quello del tremolo) in un altro dei registri del SID. Quindi la comunicazione con l'esterno serve anche a fare collegamenti tra moduli del SID che non sono previsti dal normale percorso del segnale.

Infine, ci sono due locazioni di memoria, inserite nel gruppo che controlla o legge il SID, che hanno a che fare col nostro sintetizzatore solo indirettamente. Si tratta delle locazioni 54297 e 54298, che riportano i dati di lettura delle paddles o del joystick, aggiornati circa ogni mezzo secondo. Prelevando questi dati e riversandoli in qualche registro del SID si rende possibile un controllo manuale dei parametri di funzionamento di qualcuno dei moduli.

TABELLA DEI REGISTRI DEL SID

		BIT n.	7	6	5	4	3	2	1	0
		DECIM.	128	64	32	16	8	4	2	1
INDIRIZZO		FUNZIONE	**OSCILLATORE 1**							
0	54272	LO FQ	f 7	f 6	f 5	f 4	f 3	f 2	f 1	f 0
1	54273	HI FQ	f 15	f 14	f 13	f 12	f 11	f 10	f 9	f 8
2	54274	LO PW (PL)	pw 7	pw 6	pw 5	pw 4	pw 3	pw 2	pw 1	pw 0
3	54275	HI PW (PH)	/////	/////	/////	/////	pw 11	pw 10	pw 9	pw 8
4	54276	REG. CONT.	RUM.				TEST	R MOD	SYNC	GATE
5	54277	ATT–DEC	ATT 3	ATT 2	ATT 1	ATT 0	DEC 3	DEC 2	DEC 1	DEC 0
6	54278	SOS–RIL	SOS 3	SOS 2	SOS 1	SOS 0	RIL 3	RIL 2	RIL 1	RIL 0
			OSCILLATORE 2							
7	54279	LO FQ	f 7	f 6	f 5	f 4	f 3	f 2	f 1	f 1
8	54280	HI FQ	f 15	f 14	f 13	f 12	f 11	f 10	f 9	f 8
9	54281	LO PW (PL)	pw 7	pw 6	pw 5	pw 4	pw 3	pw 2	pw 1	pw 0
10	54282	HI PW (PH)	/////	/////	/////	/////	pw 11	pw 10	pw 9	pw 8
11	54283	REG. CONT.	RUM.				TEST	R MOD	SYNC	GATE
12	54284	ATT–DEC	ATT 3	ATT 2	ATT 1	ATT 0	DEC 3	DEC 2	DEC 1	DEC 0
13	54285	SOS–RIL	SOS 3	SOS 2	SOS 1	SOS 0	RIL 3	RIL 2	RIL 1	RIL 0
			OSCILLATORE 3							
14	54286	LO FQ	f 7	f 6	f 5	f 4	f 3	f 2	f 1	f 0
15	54287	HI FQ	f 15	f 14	f 13	f 12	f 11	f 10	f 9	f 8
16	54288	LO PW (PL)	pw 7	pw 6	pw 5	pw 4	pw 3	pw 2	pw 1	pw 0
17	54289	HI PW (PH)	/////	/////	/////	/////	pw 11	pw 10	pw 9	pw 8
18	54290	REG. CONT.	RUM.				TEST	R MOD	SYNC	GATE
19	54291	ATT–DEC	ATT 3	ATT 2	ATT 1	ATT 0	DEC 3	DEC 2	DEC 1	DEC 0
20	54292	SOS–RIL	SOS 3	SOS 2	SOS 1	SOS 0	RIL 3	RIL 2	RIL 1	RIL 0
			FILTRO, VOLUME E REGISTRI DI LETTURA							
21	54293	LO TN (TL)	/////	/////	/////	/////	/////	tn 2	tn 1	tn 0
22	54294	HI TN (TH)	tn 10	tn 9	tn 8	tn 7	tn 6	tn 5	tn 4	tn 3
23	54295	RIS–FILT	RIS 3	RIS 2	RIS 1	RIS 0	FILTEX	FILT 3	FILT 2	FILT 1
24	54296	VOL–MODO	3 OFF	HP	BP	LP	VOL 3	VOL 2	VOL 1	VOL 0
25	54297	POT 1	7	6	5	4	3	2	1	0
26	54298	POT 2	7	6	5	4	3	2	1	0
27	54299	OSC 3	O 7	O 6	O 5	O 4	O 3	O 2	O 1	O 0
28	54300	ENV 3	E 7	E 6	E 5	E 4	E 3	E 2	E 1	E 0

Purtroppo ora ci avviciniamo all'unica vera difficoltà nel trattamento del SID, che richiederà un po' della vostra pazienza, a meno che non siate già dei programmatori consumati. I dati che i moduli del SID accettano come controllo non sono tutti dello stesso tipo. Alcuni dati sono a sedici bit, divisi in due bytes; altri dati sono a quattro bit, raggruppati a due a due in un solo byte; altri ancora consistono in un solo bit, e sono raggruppati otto per volta in un byte unico; come se non bastasse, ci sono altri raggruppamenti bizzarri, di dodici o perfino di undici bit. Per rendervene conto, guardate ora la tabella che rappresenta il significato di ciascuno dei bit delle ventinove locazioni di memoria (venticinque a sola scrittura, quattro a sola lettura) che ci interessano. Se non l'avete mai vista, o se vi preoccupa, non fateci caso: finirete per impararla a memoria.

Prima di affrontare il prossimo capitolo, fate girare il programma che segue. Come molti altri dello stesso tipo che sono in circolazione, vi permette di controllare i moduli del SID. Ha, però, una particolarità: tutti i valori che il programma vi richiede sono gli stessi che dovrete poi imparare a inserire nei registri del SID per farlo funzionare. Il programma ha (nelle ultime linee) delle routines di protezione per evitare che possiate inserire valori sbagliati; questo non vuol dire che sia facile tirarne fuori qualcosa! Vi consiglio di sperimentare liberamente i valori della frequenza, della modulazione dell'onda a impulsi e dell'ADSR; col filtro andateci piano (a meno che non sappiate fare i conti necessari, tabella alla mano); per iniziare, ponete MODE a 0 e VOLUME a 15, e il registro di controllo ai valori che seguono (il primo di ogni coppia fa partire l'ADSR, il secondo lo fa finire): 17-16, 33-32, 65-64, 129-128.

```
PROGRAM:  6.SINTETIZZATORE

 10 REM ******************
 20 REM **SINTETIZZATORE**
 30 REM ******************
 40 REM ***DI F. FABBRI***
 50 REM ******************
 60 PRINT "[CLEAR]":POKE 53280,5
    :POKE 53281,7
 70 PRINT "[BLACK]";CHR$(142)
    :PRINT"[DOWN]"
 80 PRINT "OSC. 1"SPC(8)"OSC. 2"SPC(8)
    "OSC. 3"
 90 CD$="[DOWN24]"
100 CL$="
      "
110 X$="              "
120 L1$="      [RED]@[BLACK,SPACE39]"
130 L2$="                        [RED]@
    [BLACK,SPACE25]"
140 L3$="
        [RED]@[BLACK,SPACE11]"
150 PRINT"[HOME]";LEFT$(CD$,20);"[RED]
    ";"COMANDI:  <1>, <2>,
     <3> (OSCILLATORE)"
```

```
160 PRINT "<F> (FREQUENZA),
    <P> (PULSE WAVE MOD)"
170 PRINT "<C> (REG. CONTR.),
    <A> (ATTACK-DECAY)"
180 PRINT "<S> (SUSTAIN-RELEASE),
    <T> (FILTRO)"
190 PRINT "<M> (MODALITA' DEL FILTRO
   E VOLUME)"
200 S=54272:FOR L=S TO S+24:POKE L,0
   :NEXT
210 GOSUB 1370
220 GOTO 1150
230 PRINT"[HOME]":PRINT CL$
240 GET U$:IF U$="" GOTO 240
250 IF U$="1" GOTO 1150
260 IF U$="2" GOTO 1180
270 IF U$="3" GOTO 1210
280 IF U$="F" GOTO 360
290 IF U$="P" GOTO 460
300 IF U$="C" GOTO 560
310 IF U$="A" GOTO 620
320 IF U$="S" GOTO 730
330 IF U$="T" GOTO 840
340 IF U$="M" GOTO 1040
350 GOTO 230
360 GOSUB 1240:INPUT "FREQ LO ";A$
   :LI=255:GOSUB 1260
370 IF ER=1 GOTO 360
380 FL=VAL(A$)
390 CD=6:Q$="FREQ LO "+A$:GOSUB 1310
400 GOSUB 1240:INPUT "FREQ HI ";A$
   :LI=255:GOSUB 1260
410 IF ER=1 GOTO 400
420 FH=VAL(A$)
430 CD=7:Q$="FREQ HI "+A$:GOSUB 1310
440 POKE S+LO,FL:POKE S+HI,FH
450 GOTO 230
460 GOSUB 1240:INPUT "PW LO ";A$
   :LI=255:GOSUB 1260
470 IF ER=1 GOTO 460
480 WL=VAL(A$)
490 CD=8:Q$="PW LO    "+A$:GOSUB 1310
500 GOSUB 1240:INPUT "PW HI ";A$:LI=15
   :GOSUB 1260
510 IF ER=1 GOTO 500
520 WH=VAL(A$)
530 CD=9:Q$="PW HI    "+A$:GOSUB 1310
540 POKE S+PL,WL:POKE S+PH,WH
550 GOTO 230
560 GOSUB 1240:INPUT "C. REG. ";A$
   :LI=255:GOSUB 1260
570 IF ER=1 GOTO 560
580 RE=VAL(A$)
590 CD=10:Q$="C. REG. "+A$:GOSUB 1310
600 POKE S+CR,RE
610 GOTO 230
620 GOSUB 1240:INPUT "ATTACK ";A$
   :LI=15:GOSUB 1260
630 IF ER=1 GOTO 620
640 AT=VAL(A$)
650 CD=11:Q$="ATTACK   "+A$:GOSUB 1310
660 GOSUB 1240:INPUT "DECAY ";A$:LI=15
   :GOSUB 1260
670 IF ER=1 GOTO 660
```

```
680 DE=VAL(A$)
690 CD=12:Q$="DECAY    "+A$:GOSUB 1310
700 A1=AT*16+DE
710 POKE S+AD,A1
720 GOTO 230
730 GOSUB 1240:INPUT "SUSTAIN ";A$
   :LI=15:GOSUB 1260
740 IF ER=1 GOTO 730
750 SU=VAL(A$)
760 CD=13:Q$="SUSTAIN "+A$:GOSUB 1310
770 GOSUB 1240:INPUT "RELEASE ";A$
   :LI=15:GOSUB 1260
780 IF ER=1 GOTO 770
790 RL=VAL(A$)
800 CD=14:Q$="RELEASE "+A$:GOSUB 1310
810 S1=SU*16+RL
820 POKE S+SR,S1
830 GOTO 230
840 GOSUB 1240:INPUT "FILT LO ";A$
   :LI=7:GOSUB 1260
850 IF ER=1 GOTO 840
860 TL=VAL(A$)
870 CD=16:SP=0:Q$="FILT LO "+A$
   :GOSUB 1340
880 GOSUB 1240:INPUT "FILT HI ";A$
   :LI=255:GOSUB 1260
890 IF ER=1 GOTO 880
900 TH=VAL(A$)
910 CD=17:SP=0:Q$="FILT HI "+A$
   :GOSUB 1340
920 GOSUB 1240:INPUT "RESON.";A$:LI=15
   :GOSUB 1260
930 IF ER=1 GOTO 920
940 RS=VAL(A$)
950 CD=16:SP=14:Q$="RESON.   "+A$
   :GOSUB 1340
960 GOSUB 1240:INPUT "SWITCH ";A$
   :LI=15:GOSUB 1260
970 IF ER=1 GOTO 960
980 SW=VAL(A$)
990 CD=17:SP=14:Q$="SWITCH   "+A$
   :GOSUB 1340
1000 R1=RS*16+SW
1010 POKE S+21,TL:POKE S+22,TH
1020 POKE S+23,R1
1030 GOTO 230
1040 GOSUB 1240:INPUT "MODE ";A$:LI=15
   :GOSUB 1260
1050 IF ER=1 GOTO 1040
1060 MO=VAL(A$)
1070 CD=16:SP=28:Q$="MODE     "+A$
   :GOSUB 1340
1080 GOSUB 1240:INPUT "VOLUME ";A$
   :LI=15:GOSUB 1260
1090 IF ER=1 GOTO 1080
1100 VO=VAL(A$)
1110 CD=17:SP=28:Q$="VOLUME   "+A$
   :GOSUB 1340
1120 V1=16*MO+VO
1130 POKE S+24,V1
1140 GOTO 230
1150 LO=0:HI=1:PL=2:PH=3:CR=4:AD=5
   :SR=6:FI=23:C%=0
1160 PRINT"[HOME]":PRINT"[DOWN2]"
   :PRINT L1$
```

```
1170 GOTO 230
1180 LO=7:HI=8:PL=9:PH=10:CR=11:AD=12
   :SR=13:FI=23:C%=14
1190 PRINT"[HOME]":PRINT"[DOWN2]"
   :PRINT L2$
1200 GOTO 230
1210 LO=14:HI=15:PL=16:PH=17:CR=18
   :AD=19:SR=20:FI=23:OF=24:C%=28
1220 PRINT"[HOME]":PRINT"[DOWN2]"
   :PRINT L3$
1230 GOTO 230
1240 PRINT"[HOME]":PRINT CL$
   :PRINT"[HOME]"
1250 RETURN
1260 ER=0:FOR J=1 TO LEN(A$)
   :J$=MID$(A$,J,1)
1270 IF J$<"0" OR J$>"9" THEN ER=1
   :J=LEN(A$):RETURN
1280 NEXT
1290 IF VAL(A$)<0 OR VAL(A$)>LI THEN
   ER=1
1300 RETURN
1310 PRINT"[HOME]";LEFT$(CD$,CD);
   SPC(C%);X$
1320 PRINT"[HOME]";LEFT$(CD$,CD);
   SPC(C%);Q$
1330 A$="":RETURN
1340 PRINT"[HOME]";LEFT$(CD$,CD);
   SPC(SP);X$
1350 PRINT"[HOME]";LEFT$(CD$,CD);
   SPC(SP);Q$
1360 A$="":RETURN
1370 PRINT"[HOME]";"[BLACK]";"[DOWN6]
   ";"FREQ LO 0";SPC(5);"FREQ LO 0";
   SPC(5);"FREQ LO 0"
1380 PRINT"FREQ HI 0";SPC(5);
   "FREQ HI 0";SPC(5);"FREQ HI 0"
1390 PRINT"PW LO   0";SPC(5);
   "PW LO   0";SPC(5);"PW LO   0"
1400 PRINT"PW HI   0";SPC(5);
   "PW HI   0";SPC(5);"PW HI   0"
1410 PRINT"C. REG. 0";SPC(5);
   "C. REG. 0";SPC(5);"C. REG. 0"
1420 PRINT"ATTACK  0";SPC(5);
   "ATTACK  0";SPC(5);"ATTACK  0"
1430 PRINT"DECAY   0";SPC(5);
   "DECAY   0";SPC(5);"DECAY   0"
1440 PRINT"SUSTAIN 0";SPC(5);
   "SUSTAIN 0";SPC(5);"SUSTAIN 0"
1450 PRINT"RELEASE 0";SPC(5);
   "RELEASE 0";SPC(5);"RELEASE 0"
1460 PRINT
1470 PRINT"FILT LO 0";SPC(5);
   "RESON.  0";SPC(5);"MODE    0"
1480 PRINT"FILT HI 0";SPC(5);
   "SWITCH  0";SPC(5);"VOLUME  0"
1490 RETURN
```

L'ORGANIZZAZIONE DEI DATI: E' ORA DI DARE I NUMERI!

Chi è già esperto di programmazione, e non ha difficoltà con il sistema binario o con quello esadecimale, può saltare al prossimo capitolo.

Gli altri non si spaventino: si tratta solo di riflettere su cose che facciamo tutti i giorni (questa, naturalmente, è la vera difficoltà).

Perché quando vediamo le tre cifre 127 tendiamo a leggere "centoventisette" e non "uno due sette"? Perchè enunciamo il risultato di una serie di operazioni che siamo stati abituati a fare : "uno per cento uguale cento, due per dieci uguale venti, sette per uno uguale sette; cento più venti più sette uguale centoventisette". In cifre: $(1 * 100)+ (2 * 10)+ (7 * 1) = 127$.

Se ricordiamo che $100 = 10^2$, $10 = 10^1$, e $1 = 10^0$ (ogni numero elevato alla prima potenza è uguale a se stesso, e ogni numero elevato alla potenza zero è uguale a uno), allora la piccola espressione (che traduce sulla carta il nostro percorso mentale di quando leggiamo il numero "centoventisette") diventa: $(1 * 10^2) + (2 * 10^1) + (7 * 10^0) = 127$.

(Notate che le parentesi sono superflue, dato che l'elevamento a potenza e la moltiplicazione hanno la precedenza sulla somma).

Quando leggiamo un numero, scritto con il sistema di numerazione decimale, moltiplichiamo ogni cifra per una potenza di dieci, che è determinata dalla posizione della cifra: la cifra più a destra (in un intero: altrimenti quella che si trova immediatamente a sinistra della virgola) va moltiplicata per dieci alla zero, poi, man mano che si procede verso sinistra, si aumenta ogni volta di uno l'esponente della potenza. Ecco un altro esempio (per un numero con la virgola, sostituita qui - dato che trattiamo di computer - con un punto): $5386.12 = 5 * 10^3 + 3 * 10^2 + 8 * 10^1 + 6 * 10^0 + 1 * 10^{-1} + 2 * 10^{-2}$. "Cinquemilatrecentoottantasei virgola dodici". Qui la cifra più a destra è moltiplicata per dieci alla meno due, cioè un centesimo. La cifra più a destra di un numero si chiama *cifra meno significativa*, quella più a sinistra *cifra più significativa*.

Ma perché usiamo le potenze di dieci? Perché il nostro sistema di numerazione usa dieci cifre: da 0 a 9.

Immaginiamo allora di avere un sistema di numerazione con due sole cifre: 0 e 1. Chiamiamo il sistema, invece che decimale, binario (si sarebbe potuto chiamare anche duale, ma non importa). Chiamiamo ognuna di queste due cifre *bit*, abbreviazione di *binary digit*, cifra binaria (aha!).

Un numero binario sarà rappresentato da una successione di cifre binarie, cioè da una successione di bit. Ad esempio: 1001. Per sapere quanto vale questo numero, dobbiamo moltiplicare ogni cifra per una potenza di due (perché due è il numero di cifre del nostro sistema), sempre cominciando da destra con l'esponente zero e aumentando di uno man mano che si va verso sinistra, e poi dobbiamo sommare i valori ottenuti: $1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 9$.

Quindi, 1001 corrisponde al numero decimale 9.

Osservate che, per quanto noiosi, questi calcoli sono semplificati dal numero ridotto di cifre del sistema, e dal loro valore: se la cifra è 0 basta ignorare la potenza di due che corrisponde alla posizione di quella cifra, se è 1 basta prendere la potenza di due tale e quale. Se ora andate a riprendere la tabella dei registri del SID, vedrete che le prime due righe in alto riportano rispettivamente: 1) i valori degli esponenti delle potenze di due; 2) i valori delle potenze. E' un aiuto per calcolare quanto vale in decimale una certa successione di cifre binarie. Infatti, basta guardare in quali colonne si trovano gli 1, e sommare le potenze di due che corrispondono a quelle colonne. Ecco un esempio:

Bit n.	7	6	5	4	3	2	1	0
Decim.	128	64	32	16	8	4	2	1
Numero	0	1	0	0	1	0	1	1

Il numero binario riportato nella riga "Numero" (01001011) vale 64 + 8 + 2 + 1 = 75. Chiaro?

Un numero binario come quello dell'esempio, costituito da otto cifre (bit), si chiama byte (lo sapevamo già). Con il sistema binario normale (ahimé, ce ne sono anche di "speciali", ma non ce ne occuperemo) il valore massimo che può essere rappresentato da un byte è 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255. Per un solo byte un valore maggiore di 255 è una ?ILLEGAL QUANTITY. Dato che ora siete in grado di convertire un numero binario in uno decimale, siete capaci di calcolare che valore bisogna POKare in una locazione per fare sì che certi bit siano uguali a 1 e certi altri uguali a 0. In computerese un bit uguale a 1 si chiama anche *alto*, uno uguale a 0 si chiama *basso*.

Ma come si fa a fare l'operazione inversa, cioè convertire un numero decimale in binario? Ovvero: se ottengo un certo risultato da un PRINT PEEK (4), come faccio a sapere quali bit della locazione 4 sono alti e quali sono bassi?

Esistono vari metodi. Vediamone uno semplicissimo. Si prende il valore (facciamo che sia 177) e si assegna un 1 alla più alta potenza di due che non supera quel numero. In questo caso è 128, cioè 2^7. Quindi il bit 7 è alto. Poi si sottrae al valore di partenza il valore della potenza di due che si è appena trovato: 177 - 128 = 49. Si ripete l'operazione: la potenza di due più alta che non supera 49 e 32, cioè 2^5. Quindi il bit 5 è alto. Poi si sottrae ancora: 49 − 32 = 17. La potenza più alta di due che non supera 17 è 16, cioè 2^4. Quindi il bit 2 è alto. Poi si sottrae ancora: 17 − 16 = 1. La potenza più alta di due che non supera 1 è 1, cioè 2^0. Quindi il bit 0 è alto. Poi si sottrae ancora: 1 − 1 = 0. Fine. Sono alti i bit 7,5,4,0. Il numero è 10110001.

Questa procedura poteva essere descritta in modo più sintetico come segue:

1) Prendi il numero;
2) trova la potenza più alta di due che non supera il numero;
3) assegna un 1 al bit che corrisponde alla potenza di due appena tro-

vata;
4) sottrai al numero la potenza appena trovata;
5) se il risultato è diverso da zero, chiamalo "numero", e ritorna a 1); altrimenti:
6) mostra un numero binario, con i bit alti trovati, e gli altri bassi. Fine.

Questa procedura, o *algoritmo*, potrebbe essere realizzata (gli informatici direbbero "implementata", usando un verbo che in Italiano non esiste: purtroppo c'è un sacco di gente che dice "ho realizzato" al posto di "ho capito", e "ho implementato" invece di "ho realizzato") con un programma per il vostro computer. In realtà, se ci si limita a numeri di un byte, è talmente facile da fare con carta e matita o a mente che non merita di occupare la memoria del Commodore 64. Potete *realizzarla* per esercizio.

Ma ci sono alcune altre cose che dovete sapere, a proposito del sistema binario.

Intanto, avrete forse notato che spostando tutte le cifre di un numero binario a sinistra di una posizione (e mettendo uno zero nel posto lasciato vuoto) si ottiene un numero che è uguale a quello di partenza moltiplicato per due. Esempio: 00001100 = 12, 00011000 = 24. E' esattamente lo stesso che succede quando si aggiunge uno zero a destra di un numero decimale: là si ottiene dieci volte tanto, qui due volte. Quindi, se vogliamo spostare mezzo byte da destra a sinistra (esempio, da xxxx1010 a 10100000, dove xxxx sono cifre che non ci interessano) dobbiamo moltiplicare quattro volte per due, cioè per sedici. Mezzo byte si chiama *nybble*. Questa tecnica ci servirà quando avremo a che fare con quei registri del SID che ospitano per metà un valore di controllo, e per un'altra metà qualcos'altro.

Un modo molto comodo di lavorare con i singoli bit è offerto dalle operazioni logiche. Nel Basic del Commodore 64 ci sono due operatori logici, AND e OR. Troverete spiegazioni più approfondite sui manuali del computer; comunque non è niente di difficile: si tratta di operatori (se la parola vi spaventa, anche "+", "−", "x" e ":" sono operatori) che confrontano due bytes bit a bit. Cioè prendono il bit zero di un byte e lo confrontano con il bit zero dell'altro, poi prendono il bit uno dell'uno e dell'altro, e così via (in realtà fanno tutti questi confronti allo stesso tempo). Tra AND e OR cambia solo il risultato del confronto: con AND il risultato è 1 solo se tutti e due i bit confrontati sono uguali a 1 (e 0 in tutti gli altri casi); con OR per avere un 1 basta che anche uno solo dei bit confrontati sia 1 (e si ha 0 solo se tutte e due i bit confrontati sono 0). Ad esempio:

```
11010010 AND        11010010 OR
01011001 =          01011001 =
.............       .............
01010000            11011011
```

Chiaro? Le istruzioni AND e OR possono essere usate per "forzare" certi bit, usando particolari bytes come *maschere*. Ad esempio, se non sappiamo che valore abbia un certo byte e vogliamo che il bit zero diventi

uguale a 0, lasciando gli altri invariati, possiamo fare una AND con la maschera 11111110. In questo modo il bit zero diventerà per forza 0, e gli altri saranno 0 se erano già 0, 1 se erano già 1. Viceversa, se vogliamo forzare a 1 il bit zero, faremo una OR con la maschera 00000001: il bit zero diventerà 1, gli altri resteranno com'erano.
Ecco gli esempi:

00010001 AND	00010000 OR
11111110 =	00000001 =
.....	.....
00010000	00010001

In uno dei primi programmi che abbiamo visto (quello sulle forme d'onda) c'era un'istruzione AND 254 (11111110), per forzare a 0 il bit zero di uno dei registri del SID. Se ora andate a vedere la tabella dei registri, vedrete che era il bit di gate. Il gate del SID 6581, invece che premendo o rilasciando un tasto, si comanda molto facilmente con una OR 1 o una AND 254.

Infine, per concludere la nostra scorribanda nel pianeta delle cifre, una domanda imbarazzante. Se un byte può contenere solamente numeri compresi tra 0 e 255, come fa il computer a trattare numeri più grandi e più piccoli? La risposta (a meno del segno e della virgola, di cui non ci occuperemo) è quasi ovvia: usando più bytes. Il computer può mettere questi bytes in locazioni successive, e assegnare a queste locazioni un valore di posto (come avviene per le cifre di un numero decimale o binario): in genere la locazione con l'indirizzo più basso ospita il byte meno significativo. Insomma, limitandoci a numeri di sedici bit, un byte (nell'indirizzo più basso) contiene i primi otto bit, e quello successivo contiene gli otto bit più significativi.

Ad esempio, il numero a sedici bit 1100110010101010 sarebbe diviso nei due bytes 11001100 e 10101010; se fossero ospitati nelle locazioni di memoria 49152 e 49153 la prima (con l'indirizzo più basso) conterrebbe 10101010, la seconda 11001100. Questa distribuzione in vari bytes viene fatta dal computer automaticamente, in base alle istruzioni del sistema operativo (con sofisticazioni, in particolare per le variabili reali, che non ci interessano direttamente). Ma in alcuni casi è necessario saper dividere un numero di più di otto bit in due bytes, o trasformare due bytes in un numero a più di otto bit. Come abbiamo visto, questo è necessario per lavorare col SID.

Il problema si può risolvere tenendo presenti due caratteristiche che abbiamo già osservato, e che si equivalgono. Quando le cifre di un numero a più di otto bit sono tutte in fila, i bit oltre l'ottavo (il bit sette, perché si comincia a contare dal bit zero) corrispondono a potenze di due maggiori di 128 (2^7): andando sempre verso sinistra, corrispondono a 256 (2^8), 512 (2^9), 1024 (2^{10}), eccetera. Ma se si spezza il numero dividendolo in due bytes, il nono bit diventerà il primo bit del byte più significativo, il decimo diventerà il secondo, eccetera. Tutti i bit vengono spostati a destra di otto posti. Per ridargli il loro vero valore, bisogna rispostarli a sinistra di otto posti. Questo si ottiene moltiplicando il byte più significativo per 2^8, cioè 256.

L'altro modo equivalente di mettere la questione è di pensare ai bytes come alle cifre di un sistema di numerazione a base 256 (256 è il numero di tutti i valori diversi che può assumere un byte). Allora un numero di più bytes può essere trasformato in un unico numero decimale moltiplicando ogni byte per l'opportuna potenza di 256.

Che la si veda in un modo o nell'altro, se il byte meno significativo è 214 e quello più significativo è 28, il numero rappresentato è 28 * 256 + 214 = 7382 (la seconda interpretazione del problema sarà chiara se si scrive così: 28 * 256^1 + 214 * 256^0 = 7382).

Viceversa, dato un numero decimale più grande di 255, come si fa a dividerlo in più bytes? Si procede così: si divide per la massima potenza di 256 che non supera il numero; se l'operazione dà un risultato intero (senza decimali dopo la virgola), questo viene assegnato al byte che corrisponde, nell'ordine, a quella potenza, e quelli meno significativi sono posti a zero. Se invece l'operazione ha resto, si assegna la parte intera del risultato al byte in questione, e si calcola il resto. Questo viene a sua volta diviso per la massima potenza di 256 che non la supera, e così via.

La procedura sembra più complicata di quella per trasformare un numero decimale in binario, ma solo perché qui si tratta di dividere anziché di sottrarre. In realtà, specialmente per noi che dobbiamo lavorare con numeri non più grandi di sedici bit, si tratta di un paio di operazioni.

Ad esempio, sia il nostro numero 12431. La massima potenza di 256 che non lo supera è 256^1. Quindi facciamo 12431 : 256 = 48.56 (approssimato alle prime due cifre dopo la virgola). Allora il byte che corrisponde a 256 è uguale a 48. Troviamo il resto: 48 * 256 = 12288; 12431 − 12288 = 143 (questo è il resto). La massima potenza di 256 che non lo supera è 256^0, cioè 1. Allora facciamo 143 : 1 = 143, e quindi il nostro compito è finito: 12431 si divide in due bytes, 48 (più significativo) e 143 (meno significativo).

A qualcuno questo modo di trovare il resto sembrerà macchinoso, ma il fatto è che ormai quasi nessuno al mondo per dividere 12431 per 256 si mette a fare: "Due in uno non ci sta; due in dodici sta sei volte, ma cinque in quattro non ci sta; due in dodici sta cinque volte, due per cinque dieci, al dodici due, cinque in ventiquattro non sta cinque volte; due in dodici sta quattro... " che è il modo più "rapido" per sapere il resto di una divisione. Si prende una calcolatrice tascabile, si fa 12431 : 256, e si ottiene 48 seguito da una serie di cifre decimali. Quindi il resto bisogna calcolarselo.

Nei nostri programmi, ogni volta che dovremo trasformare un numero rappresentabile da sedici bit nei due bytes corrispondenti, faremo fare queste operazioni direttamente al computer. Chiamando LO e HI rispettivamente il byte meno significativo ("basso") e quello più significativo ("alto"), useremo una routine di questo tipo (NU è il numero da trasformare):

```
1000 HI = INT (NU/256)
1010 LO = NU-HI * 256
```

Bene. Ora sappiamo tutto quello che ci serve per andare avanti.

LA FREQUENZA

La frequenza fondamentale del segnale generato dagli oscillatori del SID è controllata da un valore a 16 bit, diviso in due bytes (basso e alto, cioè meno significativo e più significativo). Come si può notare dalla tabella dei registri del SID, i parametri specifici di ogni oscillatore sono controllati da un gruppo di sette registri: quelli della frequenza sono i primi due all'interno di ciascun gruppo, cioè 54272 e 54273, 54279 e 54280, 54286 e 54287. Secondo la consetudine, il byte basso è il primo di ogni coppia, il byte alto il secondo.

Dunque la frequenza fondamentale viene rappresentata da valori compresi tra 0 e 65535 ($255 * 256^1 + 255 * 256^0$: il massimo valore rappresentabile con sedici bit). Questi valori, ovviamente, non coincidono con la misura in Hertz della frequenza generata: questa infatti varia soltanto fra 0 e 3906 (o, con maggior precisione, 3906.19041). Dividendo 65535 per 3906.19041 si ottiene 16.777216. Questo numero è molto utile: rappresenta il fattore per cui bisogna moltiplicare la frequenza in Hz se si vuole ottenere il valore a 16 bit che la rappresenta nel SID. Ad esempio, per sapere che valore a 16 bit rappresenta una frequenza di 440 Hz si fa: $440 * 16.777216 = 7381.97504$. Applicando a questo valore l'algoritmo che abbiamo visto alla fine del capitolo precedente, si ottiene 28 per il byte alto e 213.97504 per il byte basso. Il fatto che si ottenga un valore con la virgola per il byte basso deriva dall'algoritmo, e non è un inconveniente. Infatti, il Basic arrotonda per troncamento (cioè togliendo tutto quello che sta dopo la virgola) tutti i valori reali che vanno trattati come interi. Se il valore viene messo (come nella nostra piccola routine) in una variabile reale chiamata LO, POKE 54272,LO metterà in 54272 il valore intero 213. Un programma che volesse essere più preciso dovrebbe arrotondare non per troncamento ma al valore più vicino: è per questo che nel programma "diapason" la frequenza di 440 Hz è rappresentata con i valori 28 e 214.

Che imprecisione può portare il fatto di arrotondare per troncamento? Al massimo di un'unità (che a sua volta è un arrotondamento di 0.999999...) nel byte basso. Un'unità nel valore a 16 bit corrisponde a $3906.19041 : 65535 = 0.059604645$ (Hz). Questo valore (che è il reciproco di 16.777216) rappresenta la *risoluzione* in frequenza del SID, vale a dire la più piccola differenza di frequenza che il SID sia in grado di definire: circa sei centesimi di Hertz! Viceversa, per avere un cambiamento di 1 Hz nella frequenza generata occorrerà cambiare il valore che la rappresenta di 16.777216 unità, cioè (arrotondando al valore più vicino) di 17 unità.

Tutti questi numeri derivano da un unico valore, che è 16777216

(2^{24}): questo valore, diviso per la *frequenza di clock* del computer, dovrebbe dare il rapporto tra il numero a 16 bit e la frequenza del segnale generato. In realtà la frequenza di clock del Commodore 64 (clock è il cronometro che regola l'attività del microprocessore) è 1023000 Hz (1.023 MHz), ma valori esatti della frequenza generata si ottengono dividendo 16777216 per 1000000. Questa incongruenza è all'origine del fatto che le tabelle di valori publicate dalla Commodore per le note musicali (basate sul clock di 1.023 MHz) diano luogo a suoni calanti. Per valori in accordo con l'esperienza (e con l'orecchio) vedete l'Appendice di questo libro.

Potrete facilmente verificare che anche con un numero minore di cifre significative in questi rapporti i risultati non cambiano: usando 16.7772 al posto di 16.777216 oppure 0.059605 al posto di 0.059604645 otterrete delle variazioni nella parte decimale del valore da inserire nel byte basso, che comunque viene arrotondato; un'ulteriore riduzione delle cifre significative, invece, porterebbe effettivamente a risultati diversi.

Il programma che segue trasforma il Commodore 64 in un generatore di frequenza accordabile. La frequenza di partenza è 440 Hz (o meglio, 440.001489) e può essere aumentata o diminuita premendo i tasti f1 e f3. La regolazione può essere fine lasciando le cose come stanno o premendo il tasto F: in questo caso varia il byte basso, e la frequenza aumenta o diminuisce di sei centesimi di Hz alla volta; si può passare a una regolazione più grande premendo G e facendo variare così il byte alto. Il programma presenta sullo schermo i valori del byte alto, del byte basso e della frequenza.

```
PROGRAM:  7.GENERATORE

 10 PRINT"[CLEAR]":PRINT"[WHITE]";
    CHR$(142):POKE 53280,2
    :POKE 53281,11
 20 PRINT" PREMI <A> PER SUONARE,
     <B> PER FINIRE"
 30 PRINT:PRINT" <F1> PER AUMENTARE,
     <F3> PER DIMINUIRE"
 40 PRINT:PRINT" <G> PER LA REGOLAZION
    E GRANDE"
 50 PRINT" <F> PER LA REGOLAZIONE FINE"
 60 FOR I=54272 TO 54296:POKE I,0:NEXT
 70 K=0.059604645
 80 POKE 54296,15
 90 VA(0)=214:POKE 54272,VA(0)
 100 VA(1)=28:POKE 54273,VA(1)
 110 POKE 54277,160
 120 POKE 54278,250
 130 PRINT"[DOWN2]";" HI","  LO",
    " FREQUENZA"
```

```
140 GOSUB 330
150 GET A$:IF A$="" GOTO 150
160 IF A$="A" THEN POKE 54276,17
   :GOTO 150
170 IF A$="B" THEN POKE 54276,16
   :GOTO 150
180 IF A$="G" THEN RE=1:GOTO 150
190 IF A$="F" THEN RE=0:GOTO 150
200 IF A$=CHR$(133) THEN VA(RE)=VA(RE
   )+1:GOTO 220
210 GOTO 250
220 IF VA(RE)>255 AND RE=0 THEN VA(1)
   =VA(1)+1:VA(0)=0:GOSUB 290
230 IF RE=0 THEN POKE 54272,VA(0)
   :POKE 54273,VA(1):GOSUB 330
   :GOTO 150
240 IF RE=1 THEN GOSUB 290
   :POKE 54272,VA(0):POKE 54273,VA(1)
   :GOSUB 330:GOTO 150
250 IF A$=CHR$(134) THEN VA(RE)=VA(RE
   )-1
260 IF VA(RE)<0 AND RE=0 THEN VA(1)=V
   A(1)-1:VA(0)=255:GOSUB 310
270 IF RE=0 THEN POKE 54272,VA(0)
   :POKE 54273,VA(1):GOSUB 330
   :GOTO 150
280 IF RE=1 THEN GOSUB 310
   :POKE 54272,VA(0):POKE 54273,VA(1)
   :GOSUB 330:GOTO 150
290 IF VA(1)>255 THEN VA(1)=255
   :VA(0)=255
300 RETURN
310 IF VA(1)<0 THEN VA(1)=0:VA(0)=0
320 RETURN
330 PRINT "[HOME]";"[DOWN12]";
   "
        "
340 PRINT "[HOME]";"[DOWN12]";
350 PRINT VA(1),VA(0),
   K*(256*VA(1)+VA(0)):RETURN
```

Vagando su e giù per l'estensione del SID con questo programma, noterete che le variazioni più notevoli nell'altezza del suono avvengono alle frequenze basse: vale a dire che a parità di variazione di frequenza la sensazione di altezza cambia più decisamente alle frequenze basse che a quelle alte.

Questo è dovuto al fatto che l'unità percettiva che chiamiamo *intervallo* tra due altezze dipende non dalla differenza ma dal rapporto tra le frequenze di due suoni. L'intervallo-base per il nostro sistema musicale (come per quelli di molte culture non europee) è l'ottava, che corrisponde a un rapporto di frequenza uguale a 2. C'è un'ottava di intervallo tra due suoni rispettivamente di 55 e di 110 Hz, come pure c'è un'ottava tra due suoni di 1760 e 3520 Hz. Come vedete lo stesso intervallo corrisponde a differenze di frequenza molto diverse: 55 Hz in un caso, 1760 nell'altro. Ma il rapporto è lo stesso: 110 : 55 = 2, 3520 : 1760 = 2.

L'ottava è un intervallo piuttosto grande: per chi non è abituato a cantare è gia difficile intonarne una. Per questo le varie culture musicali hanno suddiviso l'ottava in intervalli più piccoli, individuando al suo interno una serie di *note*, cioè di suoni che hanno un rapporto di frequenza definito rispetto ad un suono più grave. I rapporti di frequenza che definiscono le varie note all'interno di un'ottava sono ovviamente minori di 2; le note che stanno a un'ottava di distanza portano lo stesso nome.

I criteri che hanno portato alla definizione delle note e delle scale musicali sono piuttosto complessi, e richiederebbero considerazioni filosofico-scientifiche, musicali, tecniche, per le quali è preferibile rimandarvi a testi specifici.

Tra le divisioni dell'ottava che si usano abitualmente la più nota è quella della cosiddetta *scala temperata*. In questo caso l'ottava è divisa in dodici intervalli identici, detti *semitoni*. Ogni semitono corrisponde quindi allo stesso rapporto di frequenza: se chiamiamo questo rapporto S, moltiplicando per S la frequenza della prima nota della scala (che chiameremo F) otteniamo la seconda nota, che avrà frequenza $F * S$. Moltiplicando ancora per S otteniamo la terza nota, che avrà frequenza $F * S * S$ cioè $F * S^2$. Quando avremo fatto quest'operazione dodici volte, otterremo un suono di frequenza $F * S^{12}$. Ma, dato che abbiamo deciso di dividere l'ottava in dodici semitoni, questa dodicesima nota risulterà essere un'ottava sopra a quella di partenza, e quindi avrà frequenza $F * 2$. Dato che $F * S^{12} = F * 2$, allora $S^{12} = 2$, e quindi S è uguale alla radice dodicesima di due. Questo valore, che si può approssimare con 1.05946309, è alla base del *temperamento equabile*, cioè alla divisione dell'ottava in dodici parti uguali. Tutti gli strumenti a tastiera sono accordati secondo il temperamento equabile, ponendo il La_4 (4 sta per il numero dell'ottava in cui si trova quel La) a 440 Hz, e tutte le altre note di conseguenza secondo la radice dodicesima di due.

Un intervallo formato da due semitoni si chiama *tono*. Una scala di sette note formata da una qualsiasi disposizione di cinque toni e due semitoni (purchè i due semitoni non siano consecutivi) si chiama *scala diatonica*. Le scale diatoniche più usate sono quelle maggiore e due delle tre scale minori (la scala minore armonica non è diatonica perchè contiene un intervallo di un tono e mezzo). Le successioni di intervalli sono quelle che si studiano alle medie: due toni, un semitono, tre toni, un semitono per la scala maggiore; un tono, un semitono, due toni, un semi-

tono, due toni per la scala minore naturale; un tono, un semitono, quattro toni, un semitono per la scala minore melodica ascendente (quella discendente è uguale a quella naturale).

Una scala diatonica però si può ottenere anche con una divisione dell'ottava diversa da quella del temperamento equabile. Un esempio è la *scala di Zarlino* (dal nome del teorico del Rinascimento che la elaborò), conosciuta anche come *scala naturale* (da non confondere con la scala minore naturale).

In questa scala le sette note sono ottenute (con un procedimento piuttosto complesso) dai rapporti di frequenza 1, 9/8, 5/4, 4/3, 3/2, 5/3, 15/8 (tutti rispetto alla nota più grave).

Questa scala viene intonata "naturalmente" dalle voci e dagli strumenti non a tastiera quando suonano insieme, perché dà luogo ad accordi più puri. Di fatto, però, non esiste uno strumento di intonazione certa che ci permetta di ascoltare questa scala, che resta così un po' un fantasma, di cui molti musicisti parlano senza averla mai sentita. Il programma che segue, oltre a presentare la scala temperata sia in forma *cromatica* (cioè con tutte le note esistenti nell'ottava) che in forma diatonica, materializza il fantasma. Oltre alle scale, premendo la barra dello spazio, si possono ascoltare le successioni di accordi costruite sulle sette note delle due scale diatoniche. Ascoltando le due scale insieme (temperata e naturale) noterete che le note sulle ottave corrispondono, mentre le altre, essendo di frequenza vicina ma non uguale, danno luogo a *battimenti*. Ascoltando gli accordi, noterete che quelli costruiti sulla scala temperata danno luogo pure a battimenti (fra la terza armonica della nota di base dell'accordo e la seconda armonica della nota più acuta), mentre questo non avviene con la scala naturale.

Nel caso delle scale il programma presenta sul video, nell'ordine: 1) il numero d'ordine della nota (a partire da un La di 27.5 Hz); 2) la frequenza; 3) i valori del byte alto e di quello basso.

Nel caso del confronto tra le due scale: 1) il numero d'ordine; 2) la frequenza della scala naturale; 3) la frequenza della scala temperata.

Nel caso degli accordi: 1) il grado della scala su cui è costruito l'accordo; 2) il rapporto di frequenza fra la nota centrale (*la terza*) e quella più grave; 3) il rapporto fra la nota più acuta (*la quinta*) e quella più grave.

```
PROGRAM:  8.SCALE

 10 REM***************
 20 REM***SCALE*******
 30 REM***************
 40 PRINT"[CLEAR]":POKE 53280,2
    :POKE 53281,11:PRINT"[GREEN]";
    CHR$(142):U$="[RVS]"
 50 S1$="SCALA CROMATICA TEMPERATA "
    :S2$="SCALA DIATONICA TEMPERATA "
 60 S3$="SCALA NATURALE (DIATONICA)"
 70 S4$="TEMPERATA + NATURALE      "
```

```
80 S5$="TRIADI TEMPERATE            "
90 S6$="TRIADI NATURALI             "
100 S7$="NIENTE ALTRO                "
110 PRINT "[DOWN4,RIGHT7]
   /^^^^^^^^^^^^^^^^^^^^\"
120 PRINT      "[RIGHT7]
   \                    /"
130 PRINT      "[RIGHT7]
   /                    \"
140 PRINT      "[RIGHT7]
   \   LE SCALE MUSICALI    /"
150 PRINT      "[RIGHT7]
   /                    \"
160 PRINT      "[RIGHT7]
   \ (ATTENDERE UN ATTIMO)  /"
170 PRINT      "[RIGHT7]
   /                    \"
180 PRINT      "[RIGHT7]
   \____________________/"
190 PRINT U$+"[DOWN4,RIGHT10]
   ***F.FABBRI 1984***"
200 DIM FO(85),FH%(85),FL%(85),FG(56),
   NH%(52),NL%(52),DH%(49),DL%(49),
   FD(49)
210 FG=27.5:FG(1)=FG*9/8:FG(2)=FG*5/4
   :FG(3)=FG*4/3:FG(4)=FG*3/2
   :FG(5)=FG*5/3
220 FG(6)=FG*15/8:FG(0)=FG
230 FOR I=7 TO 49 STEP 7:FOR K=0 TO 6
   :FG(K+I)=FG(K)*2↑INT(I/7+.1)
240 NEXT:NEXT
250 SI=54272:FOR I=SI TO SI+24
   :POKE I,0:NEXT
260 FOR K=0 TO 15:POKE SI+24,K:NEXT
270 FOR I=SI+5 TO SI+24 STEP 7
   :POKE I,0:POKE I+1,244:NEXT
280 W=33:K=.059604645:SE=2↑(1/12)
290 FOR I=0 TO 85
300 FO(I)=27.5*SE↑I
310 FF=FO(I)/K:FH%(I)=FF/256
   :FL%(I)=FF-FH%(I)*256
320 NEXT
330 GOSUB 550
340 FOR I=0 TO 51:NN=FG(I)/K
   :NH%(I)=NN/256:NL%(I)=NN-NH%(I)*256
   :NEXT
350 GR$(0)="I":GR$(1)="II"
   :GR$(2)="III":GR$(3)="IV"
   :GR$(4)="V":GR$(5)="VI"
360 GR$(6)="VII"
370 PRINT"[CLEAR]":PRINT "CHE COSA VU
   OI SENTIRE?":PRINT"[DOWN3,RIGHT8]
   1) "+U$+S1$
380 PRINT"[RIGHT8]2) "+U$+S2$
390 PRINT"[RIGHT8]3) "+U$+S3$
```

```
400 PRINT"[RIGHT8]4) "+U$+S4$
410 PRINT"[RIGHT8]5) "+U$+S5$
420 PRINT"[RIGHT8]6) "+U$+S6$
430 PRINT"[RIGHT8]7) "+U$+S7$
440 PRINT "[DOWN3]PREMI IL NUMERO COR
   RISPONDENTE,":PRINT "POI <RETURN>."
450 INPUT A:ON A GOTO 460,750,650,850,
   940,1070,1210
460 PRINT"[CLEAR]":FOR I=0 TO 85
470 F$=LEFT$(STR$(FO(I)),7)
480 PRINT I+1,F$,FH%(I),FL%(I)
490 POKE SI,FL%(I):POKE SI+1,FH%(I)
   :POKE SI+4,W
500 GET A$:IF A$="" GOTO 500
510 NEXT
520 FOR I=1 TO 1000:NEXT
530 POKE SI+4,W AND 254
540 GOTO 370
550 DI(0)=2:DI(1)=2:DI(2)=1:DI(3)=2
   :DI(4)=2:DI(5)=2:DI(6)=1
560 I=0:G=1
570 FOR S=0 TO 6
580 IF I=86 THEN S=7:GOTO 640
590 DL%(G-1)=FL%(I):DH%(G-1)=FH%(I)
   :FD(G-1)=FO(I)
600 I=I+DI(S)
610 G=G+1
620 NEXT
630 IF I<86 GOTO 570
640 RETURN
650 PRINT"[CLEAR]"
660 FOR I=0 TO 49
670 POKE SI,NL%(I):POKE SI+1,NH%(I)
   :POKE SI+4,W
680 F$=LEFT$(STR$(FG(I)),7)
690 PRINT I+1,F$,NH%(I),NL%(I)
700 GET A$:IF A$="" GOTO 700
710 NEXT
720 FOR I=1 TO 1000:NEXT
730 POKE SI+4,W AND 254
740 GOTO 370
750 PRINT"[CLEAR]"
760 FOR I=0 TO 49
770 POKE SI,DL%(I):POKE SI+1,DH%(I)
   :POKE SI+4,W
780 F$=LEFT$(STR$(FD(I)),7)
790 PRINT I+1,F$,DH%(I),DL%(I)
800 GET A$:IF A$="" GOTO 700
810 NEXT
820 FOR I=1 TO 1000:NEXT
830 POKE SI+4,W AND 254
840 GOTO 370
850 PRINT"[CLEAR]"
860 FOR I=0 TO 49
```

```
870 PRINT I+1,LEFT$(STR$(FG(I)),7),
   LEFT$(STR$(FD(I)),7)
880 POKE SI,DL%(I):POKE SI+1,DH%(I)
   :POKE SI+7,NL%(I):POKE SI+8,NH%(I)
890 POKE SI+4,W:POKE SI+11,W
900 GET A$:IF A$="" GOTO 900
910 NEXT
920 FOR I=1 TO 1000:NEXT
930 POKE SI+4,W AND 254
   :POKE SI+11,W AND 254:GOTO 370
940 PRINT"[CLEAR]"
950 FOR I=14 TO 28
960 R1$=STR$(FD(I+2)/FD(I))
   :R2$=STR$(FD(I+4)/FD(I))
970 X=I-14:IF I>20 THEN X=X-7
   :IF I>27 THEN X=X-7
980 PRINT GR$(X),LEFT$(R1$,7),
   LEFT$(R2$,7)
990 POKE SI,DL%(I):POKE SI+7,DL%(I+2)
   :POKE SI+14,DL%(I+4)
1000 POKE SI+1,DH%(I):POKE SI+8,
   DH%(I+2):POKE SI+15,DH%(I+4)
1010 POKE SI+4,W:GOSUB 1200
   :POKE SI+11,W:GOSUB 1200
   :POKE SI+18,W:GOSUB 1200
1020 GET A$:IF A$="" GOTO 1020
1030 POKE SI+4,W AND 254
   :POKE SI+11,W AND 254
   :POKE SI+18,W AND 254
1040 NEXT
1050 POKE SI+4,W AND 254
   :POKE SI+11,W AND 254
   :POKE SI+18,W AND 254
1060 FOR I=1 TO 1000:NEXT:GOTO 370
1070 PRINT"[CLEAR]"
1080 FOR I=14 TO 28
1090 R1$=STR$(FG(I+2)/FG(I))
   :R2$=STR$(FG(I+4)/FG(I))
1100 X=I-14:IF I>20 THEN X=X-7
   :IF I>27 THEN X=X-7
1110 PRINT GR$(X),LEFT$(R1$,7),
   LEFT$(R2$,7)
1120 POKE SI,NL%(I):POKE SI+7,NL%(I+2)
   :POKE SI+14,NL%(I+4)
1130 POKE SI+1,NH%(I):POKE SI+8,
   NH%(I+2):POKE SI+15,NH%(I+4)
1140 POKE SI+4,W:GOSUB 1200
   :POKE SI+11,W:GOSUB 1200
   :POKE SI+18,W:GOSUB 1200
1150 GET A$:IF A$="" GOTO 1150
1160 POKE SI+4,W AND 254
   :POKE SI+11,W AND 254
   :POKE SI+18,W AND 254
1170 NEXT
```

```
1180 POKE SI+4,W AND 254
   :POKE SI+11,W AND 254
   :POKE SI+18,W AND 254
1190 FOR I=1 TO 1000:NEXT:GOTO 370
1200 FOR J=1 TO 200:NEXT:RETURN
1210 PRINT"[CLEAR]":END
```

Il programma mostra alcune tecniche di controllo della frequenza: in particolare, mostra che si possono memorizzare i valori della frequenza delle note di un'ottava in un vettore (ad esempio, linee 210-220), e ottenere poi le note delle altre ottave moltiplicando per l'opportuna potenza di due (2 = un'ottava sopra, 4 = due ottave sopra, eccetera: v. linea 230). Se volete sperimentare una scala diatonica "di fantasia", sostituite i valori delle frazioni che si trovano nelle linee 210 e 220.

Esistono modi più o meno efficienti per memorizzare le frequenze in un programma che deve "suonare": quello di memorizzarle in quanto tali, in Hertz, è decisamente il meno efficente sia in termini di consumo di memoria che di velocità di esecuzione. Il modo in assoluto più efficiente, invece, è quello di rappresentarle nella forma byte basso, byte alto, e di porle in locazioni di memoria successive: tutti i bytes bassi in ordine in un blocco, tutti i bytes alti in ordine in un altro. Si può poi accedere a questi dati con routines in linguaggio macchina, o più semplicemente con una serie di PEEK. Ad esempio, se i bytes sono stati memorizzati nella zona di RAM a partire da 49152:
POKE 54272, PEEK (49152+B+L)
dove B e L sono *puntatori*, cioè variabili che a seconda del valore possono indicare un dato in una successione. (B indicherà un blocco di 256 locazioni, e quindi potrà valere 0, 256, 512 e così via; L indicherà la locazione, quindi potrà assumere valori compresi tra 0 e 255).

Questo sistema di memorizzare le frequenze può risultare di lettura piuttosto complessa. Nel programma sulle scale che abbiamo appena visto si è preferito ospitare i bytes delle frequenze in vettori, definendoli interi per risparmiare memoria.

IL GENERATORE DI INVILUPPO

Ognuno dei tre oscillatori del SID ha, come abbiamo visto, un proprio generatore di inviluppo, che controlla un modulatore di ampiezza. La forma dell'inviluppo è del tipo ADSR, dove attacco, decadimento e rilascio sono definiti come tempi di salita o ricaduta del livello del segnale, mentre il sostegno è un livello (o meglio, una percentuale del livello massimo, a sua volta definito dal registro del volume).

I quattro parametri dell'ADSR possono assumere valori compresi tra 0 e 15, e sono ospitati (per ciascuno degli oscillatori) in due registri di un byte. Se guardate la tabella dei registri del SID, troverete che le locazioni di memoria che controllano l'ADSR sono rispettivamente: 54277 e 54278, 54284 e 54285, 54291 e 54292.

Nella prima locazione di ogni coppia sono ospitati nei quattro bit più significativi il valore dell'attacco e nei quattro meno significativi quello del decadimento; nella seconda locazione di ogni coppia sono ospitati nei quattro bit più significativi il valore del sostegno e nei quattro meno significativi quello del rilascio. Come ormai dovrebbe esservi chiaro, per spostare a sinistra di quattro posti un valore a quattro bit (compreso quindi tra 0 e 15) basta moltiplicarlo quattro volte per due, cioè per sedici. Di conseguenza il valore da inserire nella prima locazione di ogni coppia è dato dal valore dell'attacco moltiplicato per sedici, più il valore del decadimento; mentre nella seconda locazione di ogni coppia va il valore del sostegno moltiplicato per sedici, più il valore del rilascio. Insomma, se i parametri dell'inviluppo sono memorizzati nelle variabili AT, DE, SU e RE, la definizione dell'inviluppo dell'oscillatore 1 si ottiene con le due istruzioni:

```
POKE 54277, AT * 16 + DE
POKE 54278, SU * 16 + RE
```

Ecco, ora, a cosa corrispondono i valori dell'attacco, del decadimento e del rilascio, secondo una tabella che potete trovare anche sui manuali Commodore. Nel leggere la tabella tenete conto che :

1) l'attacco è il tempo compreso fra l'istante in cui viene dato il gate e quello in cui il segnale raggiunge il livello massimo (stabilito dal controllo di volume della locazione 54296);

2) il decadimento è il tempo compreso tra l'istante in cui il segnale ha raggiunto il livello massimo (vedi sopra) e quello in cui raggiunge un nuovo livello, inferiore o tuttalpiù uguale a quello precedente;

3) questo nuovo livello altro non è che il sostegno. Il volume massimo (locazione 54296) viene diviso in sedici gradini, che ne indicano una quota. Quindi, mentre il livello stabilito dal volume è assoluto, quello del sostegno è relativo: un valore di sostegno di otto corrisponde a circa la metà del volume massimo, sia che questo sia 15 oppure 2. *Notate bene* che mentre gli altri tre parametri possono sempre e comunque avere valore zero, il sostegno può valere zero solo se attacco e decadimento sono abbastanza lunghi da permettere di sentire il suono. Se -

come accade spesso ai principianti - si dimentica di definire l'ADSR, non si ottiene dal SID alcun suono: infatti il segnale sale in due millesecondi (attacco = 0) al livello massimo, e poi precipita a zero (sostegno = 0) in sei millesecondi (decadimento = 0). Viceversa, se almeno il decadimento è lungo (dai 100 millisecondi in su) il sostegno può essere uguale a zero: si otterrà un suono fortemente percussivo, tipo xilofono o pizzicato di violino. Infine:

4) il rilascio è il tempo compreso tra l'istante in cui viene tolto il gate e quello in cui il segnale torna al livello zero. Se non viene definito un sostegno, il rilascio non ha senso.

VALORE	ATTACCO	DECADIMENTO RILASCIO
DEC	Tempo / Ciclo	Tempo / Ciclo
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

VALORI DELL'INVILUPPO

Con il programma *Sintetizzatore* che abbiamo già visto potete sperimentare i risultati di varie regolazioni del generatore di inviluppo. Se il funzionamento del gate non vi è ancora chiaro (ne parliamo nel prossimo capitolo), assegnate al registro di controllo il valore 33 (per dare il gate) e il valore 32 (per toglierlo). Ricordatevi, per prima cosa, di definire la frequenza (altrimenti otterrete l'inviluppo di un bel suono di frequenza zero) e il volume (altrimenti dopo il tempo di attacco il livello del vostro segnale salirà da zero a ... zero).

Il generatore di inviluppo permette di definire con precisione e varietà notevole i caratteri timbrici del suono. Quando si usa il SID come sintetizzatore monofonico a tre oscillatori (dando e togliendo il gate contemporaneamente a tutti e tre, alla maniera di un Minimoog), si possono dare inviluppi diversi a ognuno dei tre segnali, per simulare una caratteristica dei suoni reali. Infatti, non tutte le componenti armoniche di un suono reale hanno lo stesso inviluppo; se si accordano gli oscillatori 2 e 3 su frequenze che stanno nei rapporti 2:1 e 3:1 con quella dell'oscillatore 1, si può simulare l'insieme delle prime tre armoniche di un suono: assegnando inviluppi differenti ai tre oscillatori e dando il gate contemporaneamente si può sperimentare questo effetto.

Il breve programma che segue ne dà una dimostrazione. Noterete che la gamma di frequenze presenti cambia nelle varie fasi dell'inviluppo. La forma d'onda usata è quella triangolare. Potete modificare le linee 80 e 90 per dare inviluppi diversi.

```
PROGRAM:  9.ADSR DIVERSI

 10 PRINT"[CLEAR]":PRINT"[WHITE]";
    CHR$(142):POKE 53280,2
    :POKE 53281,11
 20 PRINT" PREMI <A> PER SUONARE,
     <B> PER FINIRE":PRINT
    :PRINT " (LA 440 HZ)"
 30 PRINT:PRINT " (PRIME TRE ARMONICHE
     CON ADSR DIVERSI)"
 40 FOR I=54272 TO 54296:POKE I,0:NEXT
 50 POKE 54296,15
 60 POKE 54272,214:POKE 54279,172
    :POKE 54286,130
 70 POKE 54273,28:POKE 54280,57
    :POKE 54287,86
 80 POKE 54277,38:POKE 54284,168
    :POKE 54291,201
 90 POKE 54278,222:POKE 54285,165
    :POKE 54292,110
100 GET A$:IF A$="" GOTO 100
110 IF A$="A" THEN POKE 54276,17
    :POKE 54283,17:POKE 54290,17
    :GOTO 100
120 IF A$="B" THEN POKE 54276,16
    :POKE 54283,16:POKE 54290,16
    :GOTO 100
130 GOTO 100
```

Lo stesso risultato, ma con qualche limite nella varietà delle soluzioni possibili, si può ottenere controllando la frequenza di intervento del filtro con il generatore di inviluppo dell'oscillatore 3 (su questo torneremo poi).

Il generatore di inviluppo del SID ha una caratteristica che vale la pena di approfondire, sopratutto se si ha intenzione di lavorare con programmi in linguaggio macchina, molto veloci.

Innanzitutto, come abbiamo visto, l'ADSR è insensibile a segnali di gate uguali a quelli già ricevuti (ricordate l'esempio del tasto di un organo?). Quindi, se si dà il gate a suono già iniziato l'inviluppo prosegue imperterrito, e non riparte certo da zero. Questa caratteristica viene sfruttata positivamente da certe routines iterative, usate da alcuni programmatori per controllare la durata di un suono. Infatti, si può imporre la condizione che il segnale di gate venga continuamente ridato all'interno di un ciclo iterativo, finchè non si è raggiunta la durata voluta; a questo punto il gate viene tolto. Il fatto che l'ADSR sia insensibile ai gates ripetuti uguali evita che una procedura di questo tipo faccia ripartire il suono ogni volta.

Proprio per questo, però, bisogna usare una certa cautela con i programmi in linguaggio macchina, le cui istruzioni vengono eseguite in pochi milionesimi di secondo. Se per separare due suoni si spegne e riaccende il gate (è la procedura obbligatoria), un programma in linguaggio macchina può essere in grado di farlo così rapidamente che il SID non fa a tempo ad accorgersi del cambiamento. Se questo avviene, non otterremo una nuova partenza dell'inviluppo ma la continuazione di quello precedente. I programmatori in linguaggio macchina sono avvertiti: inseriscano istruzioni utili (o tuttalpiù delle NOP) tra lo spegnimento del gate e la riaccensione, se vogliono essere sicuri che ogni suono riparta con il suo inviluppo.

Ma, a questo punto, è bene che guardiamo il gate più da vicino.

IL REGISTRO DI CONTROLLO: GATE, FORMA D'ONDA, EFFETTI SPECIALI

Ogni oscillatore del SID ha un registro di controllo, accessibile rispettivamente dalle locazioni 54276, 54283, 54290. In questo registro ogni bit agisce da interruttore per una funzione diversa, per cui è necessario tradurre accuratamente da binario a decimale se si vuol essere sicuri di ottenere quello che si vuole.

Il bit meno significativo (bit 0) controlla il gate. Quando questo bit è alto il segnale di gate è presente, quando è basso il gate non c'è. Come abbiamo visto, per il generatore di inviluppo è significativa la variazione del gate, non il suo stato. Così (verificate con il programma *Sintetizzatore*), se viene dato il gate prima di definire l'ADSR, non si ottiene nessun suono; se a questo punto viene definito l'ADSR, ugualmente il SID non suona, perchè prima bisogna spegnere il gate, e poi riaccenderlo. Naturalmente la presenza del gate non è sufficiente a far partire il suono, se contemporaneamente non viene definita una forma d'onda. Allo stesso modo, se per azzerare il gate si azzera tutto il registro la fase di rilascio non si ottiene, perché durante quest'ultima è ovviamente necessario che la forma d'onda rimanga definita. Le forme d'onda sono selezionate con i quattro bit più significativi (4, 5, 6 e 7).

Il bit 7 seleziona, se è posto a 1, la forma d'onda "rumore", cioè un'oscillazione pseudocasuale che produce una banda di frequenza centrata intorno al valore stabilito nei due registri appositi. Non è, quindi, un *rumore bianco* (l'insieme di tutte le frequenze udibili, di pari intensità), ma un *rumore colorato*, molto utile perché cambiando le frequenza centrale si possono ottenere effetti diversi senza dover agire sul filtro. Il rumore non può essere combinato con nessuna delle altre forme d'onda, quindi se il bit 7 è alto i bit 4, 5 e 6 devono essere per forza bassi.

Il bit 6 seleziona un'onda a impulsi. Vedremo poco avanti che l'ampiezza degli impulsi può essere modulata, intervenendo su due registri per ogni oscillatore. Vedremo anche che poiché un valore di zero in quei registri determina un segnale costante (non un'oscillazione), se si seleziona questa forma d'onda senza averne prima definito il grado di modulazione non si otterrà nessun suono.

Il bit 5 seleziona un'onda a dente di sega, e il bit 4 un'onda triangolare. I bit 4, 5 e 6 possono essere anche posti a 1 contemporaneamente: in questo caso il livello istantaneo dell'oscillazione è il risultato di una AND (non di una somma) tra i valori a 8 bit che esprimono i livelli istantanei delle tre forme d'onda (in ogni momento).

Il bit 3 si chiama *test*, perché può essere usato in fase di conrollo di un programma; semplicemente, "stacca" l'oscillatore ogni volta che è messo a 1, "congelandolo" nello stato in cui si trova.

Tornando alle forme d'onda, abbiamo visto che c'è una coppia di registri per oscillatore che controlla la modulazione dell'onda a impulsi. Il grado di modulazione di un'onda a impulsi viene definito come una percentuale, che indica quanta parte del tempo di un ciclo viene trascorsa sui valori positivi; il complemento a cento indica ovviamente la percentuale di ciclo trascorsa su valori negativi. Una percentuale del 50 per cento determina un'onda quadra.

La percentuale di modulazione non è rappresentata come tale, ma con un valore a 12 bit contenuto in due registri, rispettivamente 54274 e 54275, 54281 e 54282, 54288 e 54289. Dato che il valore massimo esprimibile con 12 bit è 4095, il grado di modulazione è dato dal valore a 12 bit diviso 4095 per 100, cioè dal valore a 12 bit diviso 40.95. Viceversa, dato un grado di modulazione PW desiderato, lo si potrà ottenere con queste istruzioni:

PH = INT (PW * 40.95/256)
PL = PW * 40.95 − PH * 256

dove PH e PL sono i valori da inserire rispettivamente nel byte alto e in quello basso. L'onda quadra corrisponde (circa) a un valore di 8 per il byte alto e di 0 per il byte basso. Il valore minimo e quello massimo corrispondono a un segnale che si mantiene per il 100 per cento del tempo di un ciclo sullo stesso livello, cioè a una corrente continua. Quindi se i due registri non vengono riempiti con un valore a 12 bit diverso da zero, l'onda a impulsi non produce alcun suono .

Il bit 2 controlla la modulazione ad anello tra l'oscillatore a cui appartiene quel registro di controllo e quello precedente (se l'oscillatore a cui appartiene il registro di controllo è il numero 1, allora tra questo e il numero 3: insomma, le coppie sono 1-3, 2-1, 3-2). Per ottenere la modulazione ad anello è necessario selezionare la forma d'onda triangolare, insieme al bit 2, sul primo oscillatore della coppia; dei parametri del secondo oscillatore interessa solo la frequenza. Quindi la modulazione ad anello tra l'oscillatore 2 e l'oscillatore 1, per esempio, si ottiene inserendo il valore 21 (16 per l'onda triangolare + 4 per la modulazione ad anello + 1 per il gate) nella locazione 54283. Dell'oscillatore 2 devono essere definite anche la frequenza e l'ADSR, dell'oscillatore 1 solo la frequenza.

Il bit 1 controlla, con le stesse modalità di accoppiamento appena viste per la modulazione ad anello, la sincronizzazione di due oscillatori. Vale a dire che il ciclo del primo oscillatore della coppia viene fatto ripartire dall'inizio ogni volta che parte un ciclo del secondo oscillatore. Se la frequenza del secondo oscillatore è maggiore, allora semplicemente il primo oscillatore viene costretto a vibrare sulla stessa frequenza; se invece la frequenza del secondo oscillatore è minore, il primo oscillatore manterrà la propria frequenza, ma ci sarà un brusco cambiamento della forma d'onda e uno spostamento di *fase* (detto grossolanamente: il punto dell'oscillazione a cui si è arrivati) ogni volta che il secondo oscillatore inizia il suo ciclo. Questo dà luogo a modificazioni timbriche molto interessanti.

I quattro bit significativi del byte alto non vengono usati; quindi se per errore si inserisce in questo byte un valore maggiore di 15, il controllo funzionerà ugualmente, ma si baserà esclusivamente sulla configurazione dei quattro bit meno significativi. Ad esempio, contando da 0 a 255 con un ciclo FOR... NEXT, e assegnando questo valore al byte alto della modulazione dell'onda a impulsi, si otterrà lo stesso risultato che contando sedici volte da 0 a 15 . Infatti, se ci fate caso, andando da 0 a 255 i primi quattro bit assumono la stessa configurazione sedici volte.

FILTRO E CONTROLLO DEL VOLUME

Esauriti con il capitolo precedente i registri individuali di ogni oscillatore, vediamo adesso i quattro registri del SID che controllano il filtro e il livello di uscita. Sono accessibili dalle locazioni 54293, 54294, 54295, 54296.

Il contenuto delle prime due locazioni determina la frequenza di intervento del filtro. Il fatto bizzarro è che la frequenza di intervento sia rappresentata con un numero a 11 bit, dei quali tre (quelli meno significativi) nella locazione 54293, e gli altri otto nella locazione 54294. Il bit meno significativo della locazione 54294, comunque, è in realtà il quarto del numero a 11 bit; quindi per ottenere il valore del numero che rappresenta la frequenza di intervento occorre moltiplicare il byte alto solo per 8, non (come negli altri casi) per 256. Se TN è questo numero e TH e TL sono rispettivamente i valori delle locazioni 54293 e 54294, allora:

TN = TH * 8 + TL

Il valore massimo rappresentabile con 11 bit è 2047; poichè il campo di intervento del filtro varia tra 0 e 3906 Hz, se FT è la frequenza di intervento desiderata, allora TN si ottiene così:

TN = FT * 0.52403999

dove 0.52403999 è il rapporto tra 2047 e 3906.19041. Ottenuto TN, i valori da inserire nelle due locazioni di controllo si ottengono facilmente:

TH = INT (TN/8)

TL = TN − TH* 8

La locazione di memoria 54295 contiene nei quattro bit più significativi il valore della risonanza applicata alle frequenze vicine a quella di intervento; i quattro bit meno significativi, invece, sono deviatori (*switch*): quando sono posti a 1 inviano al filtro il segnale corrispondente, altrimenti lo inviano direttamente all'uscita. Gli accoppiamenti tra bit-deviatori e segnale controllato sono i seguenti: bit 0-oscillatore 1, bit 1-oscillatore 2, bit 2-oscillatore 3, bit 3-segnale esterno (proveniente dall'ingresso EXT IN). In questo modo è possibile filtrare solo i segnali desiderati, anche se la modalità resta la stessa per tutti i segnali che passano nel filtro.

Il registro 54296 ha una struttura simile. Qui sono i quattro bit meno significativi a contenere il valore di una variabile, che è il livello generale di uscita (volume), mentre i quattro bit più significativi sono deviatori. Il bit più significativo ha una funzione molto interessante, e unica: se è alto impedisce al segnale dell'oscillatore 3 di presentarsi all'uscita. Questo

può servire ogni volta che si usa l'oscillatore 3 o il suo generatore di inviluppo per controllare i parametri di altri oscillatori; ad esempio, se l'oscillatore 3 è usato come un LFO a 16 Hz per generare effetti di tremolo, non è certo desiderabile ascoltare anche il suo segnale: il bit 7 della locazione 54296 permette di eliminarlo dall'uscita.

I bit 6, 5 e 4, invece, determinano la modalità di funzionamento del filtro. Sulla tabella sono indicati con HP, BP e LP, acronimi di High Pass (passa alto), Band Pass (passa banda) e Low Pass (passa basso). Mettendo a 1 ognuno di questi si ottiene la modalità corrispondente. Se si selezionano insieme il bit 6 e il bit 4 (HP e LP) si ottiene la modalità mancante, cioè la reiezione di banda (Band Reject, BR).

E' abbastanza comune, quando si inizia a programmare il SID, dimenticarsi di stabilire la modalità; in questo caso il filtro, ovviamente, non funziona.

Il programma che segue vi mette a disposizione un filtro programmabile. Il suono sul quale agisce è un insieme di tre "armoniche" di un La 440 Hz; potete modificarlo cambiando i valori delle frequenze dei tre oscillatori nelle linee 130 e 140.

```
10 PRINT"[CLEAR]":PRINT"[WHITE]";
   CHR$(142):POKE 53280,2
   :POKE 53281,11
20 PRINT" PREMI <A> PER SUONARE,
    <E> PER FINIRE"
30 PRINT:PRINT" <F1> PER AUMENTARE,
    <F3> PER DIMINUIRE"
40 PRINT:PRINT" <G> PER LA REGOLAZION
   E GRANDE"
50 PRINT" <F> PER LA REGOLAZIONE FINE"
60 PRINT:PRINT" <H> PER PASSA-ALTO <H
   P>"
70 PRINT" <L> PER PASSA-BASSO <LP>"
80 PRINT" <B> PER PASSA-BANDA <BP>"
90 PRINT" <R> PER REIEZIONE DI BANDA
   <BR>"
100 PRINT:PRINT" <F5> E <F7> PER LA R
   ISONANZA"
110 FOR I=54272 TO 54296:POKE I,0:NEXT
120 K=0.52403999
130 POKE 54272,214:POKE 54273,28
   :POKE 54279,172:POKE 54280,57
140 POKE 54286,130:POKE 54287,86
150 POKE 54296,15+64:Q$="HP":RS=0
160 POKE 54277,160:POKE 54284,160
   :POKE 54291,160
170 POKE 54278,250:POKE 54285,250
   :POKE 54292,250
180 VA(0)=7:POKE 54293,VA(0)
190 VA(1)=28:POKE 54294,VA(1)
200 RE=1
210 PRINT"[DOWN2]";" HI"," LO",
   " FREQUENZA INTERV."
220 GOSUB 470:GOSUB 500:GOSUB 520
230 GET A$:IF A$="" GOTO 230
240 IF A$="A" THEN POKE 54276,17
   :POKE 54283,17:POKE 54290,17
   :GOTO 230
```

```
250 IF A$="E" THEN POKE 54276,16
   :POKE 54283,16:POKE 54290,16
   :GOTO 230
260 IF A$="G" THEN RE=1:GOTO 230
270 IF A$="F" THEN RE=0:GOTO 230
280 IF A$="H" THEN POKE 54296,64+15
   :Q$="HP":GOSUB 500:GOTO 230
290 IF A$="B" THEN POKE 54296,32+15
   :Q$="BP":GOSUB 500:GOTO 230
300 IF A$="L" THEN POKE 54296,16+15
   :Q$="LP":GOSUB 500:GOTO 230
310 IF A$="R" THEN POKE 54296,80+15
   :Q$="BR":GOSUB 500:GOTO 230
320 IF A$=CHR$(135) THEN RS=RS+1
   :GOSUB 550:GOSUB 520:GOTO 230
330 IF A$=CHR$(136) THEN RS=RS-1
   :GOSUB 570:GOSUB 520:GOTO 230
340 IF A$=CHR$(133) THEN VA(RE)=VA(RE
   )+1:GOTO 360
350 GOTO 390
360 IF VA(RE)>7 AND RE=0 THEN VA(1)=V
   A(1)+1:VA(0)=0:GOSUB 430
370 IF RE=0 THEN POKE 54293,VA(0)
   :POKE 54294,VA(1):GOSUB 470
   :GOTO 230
380 IF RE=1 THEN GOSUB 430
   :POKE 54293,VA(0):POKE 54294,VA(1)
   :GOSUB 470:GOTO 230
390 IF A$=CHR$(134) THEN VA(RE)=VA(RE
   )-1
400 IF VA(RE)<0 AND RE=0 THEN VA(1)=V
   A(1)-1:VA(0)=7:GOSUB 450
410 IF RE=0 THEN POKE 54293,VA(0)
   :POKE 54294,VA(1):GOSUB 470
   :GOTO 230
420 IF RE=1 THEN GOSUB 450
   :POKE 54293,VA(0):POKE 54294,VA(1)
   :GOSUB 470:GOTO 230
430 IF VA(1)>255 THEN VA(1)=255
   :VA(0)=7
440 RETURN
450 IF VA(1)<0 THEN VA(1)=0:VA(0)=0
460 RETURN
470 PRINT "[HOME]";"[DOWN18]";
   "
         "
480 PRINT "[HOME]";"[DOWN18]";
490 PRINT VA(1),VA(0),
   (VA(1)*8+VA(0))/K:RETURN
500 PRINT"[HOME]";"[DOWN22]";
   " MODALITA':    ";Q$
510 RETURN
520 PRINT"[HOME]";"[DOWN23]";
   "                    .    "
530 PRINT"[HOME]";"[DOWN23]";
   " RISONANZA:   ";RS
540 POKE 54295,16*RS+7:RETURN
550 IF RS>15 THEN RS=15
560 RETURN
570 IF RS<0 THEN RS=0
580 RETURN
```

I REGISTRI A SOLA LETTURA

Le locazioni da 54297 a 54300 permettono di leggere dati interessanti, che possono essere usati per controllare altre funzioni del SID.

Le locazioni 54297 e 54298 contengono i valori di lettura delle paddles, aggiornati circa ogni mezzo secondo. Con le due linee di programma che seguono, per esempio, si possono controllare dalle paddles il livello di uscita e la frequenza di intervento del filtro, realizzando ("implementando"?) due bei regolatori di "Volume" e di "Tono" per il nostro strumento:

```
POKE 54295,7 : POKE 54296, (PEEK (54297)/16) OR 16
POKE 54294, PEEK (54298)
```

Le due locazioni 54299 e 54300 informano rispettivamente sullo stato dell'oscillatore 3 e del suo generatore di inviluppo. Per capire meglio cosa si deve intendere per "stato" è necessaria una piccola digressione.

La generazione digitale del suono avviene comunicando a intervalli di tempo uguali (che determinano una certa *frequenza di campionamento*) un livello di segnale (un *campione*); il dispositivo che riceve questa informazione si chiama *convertitore digitale/analogico*, perchè sulla base di un numero (il campione che ha ricevuto) genera una tensione, che è una grandezza che varia con continuità. Passato l'intervallo di tempo stabilito dalla frequenza di campionamento, il convertitore riceve un altro campione, e aggiorna il valore della tensione generata. L'onda, quindi, viene rappresentata da una successione di numeri, che indicano il valore della tensione generata dal convertitore nei successivi istanti. La precisione della rappresentazione digitale di un'onda complessa dipende dal numero di valori diversi con cui il convertitore può leggere il segnale in forma numerica: quindi dipende dal numero di bit. Un'imprecisione nella rappresentazione digitale viene tradotta in un rumore, che si chiama *rumore di quantizzazione*. Esiste una semplice formula dalla quale si può ricavare il rapporto in dB tra segnale e rumore di quantizzazione, dato il numero di bit del covertitore: $S/R = 6 \times N$ (dove S è il livello del segnale, R quello del rumore, N il numero di bit). Se ricordate, avevamo visto che il rapporto segnale/rumore del nostro SID è 48 dB: sostituendo nella formula (ammesso che non lo sapessimo già) troviamo che il numero di bit è 8.

Ciò che il SID ci fa leggere nella locazione 54299 non è altro che il valore del campione inviato dall'oscillatore 3 (prima che sia stato elaborato dal modulatore di ampiezza), nell'istante in cui abbiamo fatto la lettura. E' ovvio che per seguire l'onda nel suo svolgersi dovremmo leggere alla stessa velocità del convertitore, cioè con frequenza pari alla frequenza di campionamento. Questo non è possibile, anche se, con routines in linguaggio macchina e, per frequenze basse, anche in Basic,

siamo in grado di cogliere il segnale a intervalli abbastanza brevi da poterne ricostruire la forma.

La stessa forma dell'onda, del resto, determina i valori che potremo ottenere dalla nostra lettura: è chiaro che un'onda quadra ci fornirà in alternativa solo 0 e 255 , e che un rumore fornirà valori casuali compresi tra 0 e 255. Quest'ultimo è un sistema eccellente per ottenere valori casuali delle dimensioni di un byte senza ricorrere all'istruzione Basic RND (x).

Anche la curva del generatore di inviluppo viene rappresentata allo stesso modo, e i suoi livelli istantanei sono leggibili nella locazione 54300. Come promesso, ecco un programma che fa controllare la frequenza di intervento del filtro al generatore di inviluppo dell'oscillatore 3.

```
PROGRAM:  11.FILTER ENV.

 10 PRINT"[CLEAR]":PRINT"[WHITE]";
    CHR$(142):POKE 53280,2
    :POKE 53281,11
 20 PRINT" PREMI <A> PER SUONARE,
     <B> PER FINIRE"
 30 PRINT:PRINT" (FILTRO CONTROLLATO D
    A ENV 3)"
 40 FOR I=54272 TO 54296:POKE I,0:NEXT
 50 I=49152
 60 READ A:IF A=256 GOTO 80
 70 POKE I,A:I=I+1:GOTO 60
 80 POKE 49180,28:POKE 49181,22
 90 SYS 49152
100 POKE 54295,1+16*15
    :POKE 54296,15+16+128
110 POKE 54272,214
120 POKE 54273,28
130 POKE 54277,192:POKE 54291,201
140 POKE 54278,250:POKE 54292,204
150 GET A$:IF A$="" GOTO 150
160 IF A$="A" THEN POKE 54276,33
    :POKE 54290,1:GOTO 150
170 IF A$="B" THEN POKE 54276,32
    :POKE 54290,0:GOTO 150
180 GOTO 150
```

L'effetto che si ottiene è simile a un wah-wah, ma cambiando la modalità del filtro (linea 100: il contenuto della POKE 54296) si può avere anche l'inverso (haw-haw?). L'inviluppo dell'oscillatore 3 è controllato nelle linee 130 e 140; provate a modificarlo. Come vedete, dato che dell'oscillatore 3 serve solo l'inviluppo, basta dare e togliere soltanto il gate, senza definire né la forma d'onda (linee 170 e 180: POKE 54290, 1 e POKE 54290, 0) nè la frequenza; comunque, per essere prudenti, l'oscillatore 3 è escluso dall'uscita.

I programmatori esperti avranno riconosciuto, dalla presenza dei DATA e dal loro valore, che il programma si serve di una routine in linguaggio macchina, del resto chiamata esplicitamente dall'istruzione

SYS 49152. La routine viene caricata in memoria dalle linee 50, 60 e 70: i numeri di linea dei DATA corrispondono all'indirizzo del primo valore nella linea di DATA. La prima parte della routine serve solo a modificare l'indirizzo della routine di IRQ, il sottoprogramma al quale il Commodore 64 salta automaticamente ogni sessantesimo di secondo (principalmente per leggere la tastiera). La seconda parte è un'aggiunta in testa alla routine di IRQ, che (ogni sessantesimo di secondo, e senza interferire nel programma in corso) legge il contenuto di una locazione di memoria e lo scrive in un altra. Gli indirizzi di queste locazioni di memoria sono contenuti a loro volta nelle locazioni 49180 e 49181, in forma relativa. Ognuno di questi due indirizzi, cioè, è espresso come la differenza fra l'indirizzo in questione e l'indirizzo di partenza del SID, che è 54272. Ad esempio, l'indirizzo 54300 è espresso dal numero 54300 − 54272 = 28 (questi valori li trovate nella prima colonna a sinistra della tabella dei registri del SID).

Questa caratteristica vi permette di usare la stessa routine per leggere e scrivere in altre locazioni: basta cambiare i contenuti delle POKE 49180 e POKE 49181 (che in questo programma si trovano alla linea 80). Ricordate che in 49180 si trova l'indirizzo relativo della locazione dalla quale si legge, e in 49181 quello della locazione in cui si scrive. Ad esempio, se volete ottenere una modulazione di frequenza tra l'oscillatore 3 e l'oscillatore 1, potete fare POKE 49180,27 : POKE 48181,0. In questo modo l'oscillatore 3 controlla il byte basso della frequenza dell'oscillatore 1. Naturalmente dovete modificare il resto del programma, assegnando una frequenza e una forma d'onda all'oscillatore 3.

Per chi conosce il linguaggio Assembly, ecco il listato della piccola routine.

```
add.   data      source code
0                10 PRT
0                20 SYM
0                30 LEGGE=49180
0                40 SCRIVE=49181
0                50 ORG$C000
C000   78        60 SEI
C001   A90D      70 LDA#$0D
C003   8D1403    80 STA 788
C006   A9C0      90 LDA#$C0
C008   8D1503    100 STA 789
C00B   58         110 CLI
C00C   60         120 RTS
C00D   AC1CC0     130 LDY LEGGE
C010   AE1DC0     140 LDX SCRIVE
C013   B900D4     150 LDA 54272.Y
C016   9D00D4     160 STA 54272.X
C019   4C31EA     170 JMP$EA31
```

L'ORGANIZZAZIONE DEI SUONI NEL TEMPO

Fare musica vuol dire, tra l'altro, organizzare suoni nello spazio e nel tempo. Fino a questo punto, quindi, non abbiamo avuto molte occasioni di fare musica, almeno intenzionalmente (cosa ne pensino i vicini o le persone che vivono con noi è un altro discorso): piuttosto, abbiamo fatto degli esperimenti di elettroacustica.

Ma il Commodore 64 ci offre anche la possibilità di organizzare suoni nel tempo (quanto allo spazio, la sua uscita mono non permette molte soddisfazioni). Questa possibilità nasce dallo stesso funzionamento del computer, che non è (né potrà mai essere) immediato. Il Commodore 64, come ogni computer, impiega del tempo a eseguire le nostre istruzioni: da pochi milionesimi di secondo, se si tratta di un'istruzione in linguaggio macchina, a numerosi secondi, se si tratta di un'istruzione complessa in Basic (come un ciclo FOR... NEXT). Quindi, se si riesce a stabilire con precisione quanto tempo il Commodore 64 impiega a eseguire un certo insieme di istruzioni, si può usare questo insieme per separare due eventi sonori successivi.

La soluzione più usata da molti programmatori è quella classica con cui si tratta la durata degli eventi nelle applicazioni più svariate, dai videogiochi elementari ai programmi gestionali: il ciclo FOR... NEXT a vuoto. Funziona così: con un'istruzione FOR... TO... STEP... viene creata una variabile destinata ad essere incrementata o decrementata (di una quantità indicata dall'argomento di STEP, e secondo il suo segno) dalla successiva istruzione NEXT, fino a che non abbia raggiunto il valore finale specificato dall'argomento di TO. Ad esempio: FOR I=255 TO 0 STEP -1:NEXT. Come è noto questo insieme di istruzioni viene usato di solito per far eseguire al computer più volte la stessa operazione (o un insieme di operazioni). Per esempio FOR K=1 TO 100:PRINT "ECCO!":NEXT fa stampare cento volte di seguito la scritta ECCO!. Se si toglie l'insieme di istruzioni intermedie (nell'esempio, PRINT "ECCO!") il computer ripete un certo numero di volte... niente: ma questo non vuol dire che non faccia niente. Infatti: 1) crea la variabile che funge da contatore (cioè riserva uno spazio in memoria per K, o I, o come l'abbiamo chiamato); 2) confronta con il valore finale indicato da TO; 3) se il valore raggiunto dal contatore è maggiore di quello finale (o minore se si sta contando alla rovescia, avendo indicato uno STEP negativo), salta all'istruzione che segue NEXT, cioè esce dal ciclo; 4) altrimenti, esegue ciò che segue FOR... TO... e che precede NEXT (nel caso che stiamo discutendo, niente); 5) aumenta o diminuisce il contatore della quantità indicata da STEP e secondo il suo segno (se, come nel nostro caso, STEP non è specificato, aumenta il contatore di 1); 6) torna al punto 2.

Tutte queste operazioni richiedono un certo tempo, per cui un ciclo

FOR... NEXT vuoto inserisce un ritardo nel programma, che dipende principalmente dal valore finale specificato da TO (assumendo di partire da zero e di non indicare lo STEP).

Molti programmi musicali, quindi, per specificare la durata di una nota o di una pausa inseriscono cicli vuoti tra i momenti di inizio dei due eventi (fatti coincidere con le rispettive istruzioni); empiricamente si trovano i valori dell'argomento di TO che corrispondono a varie durate, e li si sostituiscono ai valori musicali.

Il programma che segue è costruito in questo modo "classico", ed esegue una breve melodia. Altezze e durate sono memorizzate nei DATA a terne (byte alto e byte basso della frequenza, valore finale del ciclo FOR... NEXT vuoto); notate che terminato un suono il gate viene azzerato (linea 100): altrimenti il programma non sarebbe in grado di eseguire note ribattute (che comunque in questo caso non ci sono).

```
PROGRAM:  12.RAINBOW 1

 10 PRINT"[CLEAR]":PRINT"[WHITE]";
    CHR$(142):POKE 53280,2
    :POKE 53281,11
 20 FOR I=54272 TO 54296:POKE I,0:NEXT
 30 POKE 54277,35:POKE 54278,199
 40 POKE 54296,15
 50 PRINT " (UNA MELODIA DI [RVS]DATA
    [RVOFF])"
 60 READ HF,LF,DU
 70 IF DU=-1 GOTO 120
 80 POKE 54272,LF:POKE 54273,HF
    :POKE 54276,33
 90 FOR I=0 TO DU:NEXT
100 POKE 54276,32
110 GOTO 60
120 PRINT:PRINT " PREMI UN TASTO PER
    SENTIRLA ANCORA"
130 GET A$:IF A$="" GOTO 130
140 PRINT"[CLEAR]":RESTORE:GOTO 50
150 DATA 17,37,1200,34,75,1200,32,94,
    270
160 DATA 25,177,580,28,214,270,32,94,
    580
170 DATA 34,75,580,17,37,1200,28,214,
    1200
180 DATA 25,177,2380,-1,-1,-1
```

Volendo, potete provare a continuare la melodia, trovando i valori per le note che mancano e aggiungendoli in altre linee di DATA (e ricordandovi di spostare i tre -1, che segnalano la fine del pezzo). Ma come calcolare i valori da assegnare per le durate?

Il programmino che segue vi sarà sempre utile se vorrete usare i cicli FOR... NEXT vuoti: serve a calcolare quale ritardo si ottiene attribuendo valori diversi a K nell'espressione FOR I = 0 TO K : NEXT.

```
PROGRAM:  13.FOR...NEXT

 10 PRINT"[CLEAR]":PRINT"[WHITE]";
    CHR$(142):POKE 53280,2
    :POKE 53281,11
 20 INPUT " VALORE";K:PRINT
    :PRINT " VALORE","DURATA":PRINT
 30 GOSUB 90
 40 PRINT K,DU
 50 OPEN 3,4:CMD 3:PRINT K,DU:PRINT#3
    :CLOSE 3
 60 PRINT:PRINT " PREMI UN TASTO PER U
    N ALTRO VALORE"
 70 GET A$:IF A$="" GOTO 70
 80 PRINT"[CLEAR]":GOTO 20
 90 POKE 160,0:POKE 161,0:POKE 162,0
100 FOR I=0 TO K:NEXT
110 C=PEEK(162):D=PEEK(161)
    :E=PEEK(160)
120 DU=(C+D*256+E*4096)/60
130 RETURN
```

Se non avete una stampante, semplicemente togliete la linea 50 (e fornitevi di carta e matita!).

Facendo girare il programma e provando con diversi valori di K, è facile vedere che il ricorso ai cicli vuoti non dà molte garanzie di musicalità, perché i valori da inserire non sono proporzionali alle durate (creando problemi nel caso che si voglia cambiare il tempo metronomico); né il sistema è particolarmente efficiente da un punto di vista informatico, visto che durate anche brevi richiedono valori a più cifre e occupano molto spazio in memoria. Vediamo le questioni separatamente. Un valore di 200 (FOR I=0 TO 200) dà un ritardo di un quarto di secondo: andrebbe bene per un ottavo con un metronomo a 120, o per un sedicesimo con un metronomo a 60 (il metronomo, lo ricordo, indica il numero di unità di durata che si succedono in un minuto, dove in genere l'unità è il quarto, o semiminima). Ma se si raddoppia il valore, portandolo a 400, il ritardo è solo di 0.45 secondi; ciò significa che per avere una nota della durata doppia occorrerà un valore più che doppio (450 va bene); in questo modo, però, se si vuol dimezzare il metronomo è necessario fornire al programma serie diverse di valori finali del ciclo FOR... NEXT (sotto forma di DATA), perché non è corretto (come sarebbe invece se tra valori e durata esistesse una proporzionalità) moltiplicare o dividere i valori per una costante. Inoltre, si vede che per indicare una durata di appena un quarto di secondo (venticinque centesimi, o, come vedremo, quindici sessantesimi) sono necessarie tre cifre: se se ne potesse risparmiare solo una, ogni cento note avremmo cento bytes in meno nei nostri DATA. Non sarebbe conveniente?

Certo, e la possibilità esiste.

Il Commodore 64 ha un cronometro interno, che è sempre attivo durante il funzionamento normale del computer (si ferma solo durante le operazioni di input/output su nastro). La risoluzione del cronometro è di un sessantesimo di secondo, che è piu che sufficiente per moltissimi impieghi musicali. Il valore del conteggio è sempre contenuto nella variabile TI$ (dove compare come ore, minuti, secondi, nella forma hhmmss, ad esempio 171636 per indicare 17 ore 16 minuti 36 secondi) e in quella TI (dove compare in sessantesimi di secondo, ad esempio 3600 per indicare un minuto). Entrambe le variabili si aggiornano sulla base del contenuto di tre locazioni di memoria della pagina zero (160, 161 e 162: "pagina zero" significa semplicemente le prime 256 locazioni di memoria), nelle quali è registrato il valore del conteggio diviso in tre bytes: quello più significativo si trova in 160, quello meno significativo in 162. L'espressione PEEK(160)*4096 + PEEK(161)*256 + PEEK(162) fornisce il valore di TI. Il valore contenuto nelle tre locazioni viene aggiornato all'inizio di ogni routine IRQ.

In particolare, la locazione 162 contiene un valore in sessantesimi di secondo, che varia da 0 a 255: il valore massimo, quindi, supera i quattro secondi (255/60=4.25), ed è più che sufficiente per la maggior parte degli impieghi musicali. Dato che queste locazioni possono essere azzerate con una semplice istruzione (es. POKE 162,0), forniscono un cronometro utilissimo per fare musica.

Come si fa? Semplice. Al posto del ciclo FOR... NEXT vuoto, si inserisce (subito dopo l'istruzione che fa iniziare l'evento sonoro) una POKE 162,0 per azzerare il cronometro. Poi si dice al computer di aspettare finché il valore contenuto in 162 non abbia raggiunto la durata prestabilita per l'evento, al termine della quale si prosegue.

Il vantaggio di questo sistema è che le durate sono rappresentate da valori più piccoli, e soprattutto che questi valori sono perfettamente proporzionali. In questo modo, per cambiare il metronomo di un pezzo, è sufficiente moltiplicare o dividere i valori delle durate per una costante. E' quello che avviene nel programma seguente: non fa altro che ripetere la stessa melodia che abbiamo già sentito, ma offrendo la possibilità di riascoltarla con metronomi differenti. I valori delle durate nei DATA sono espressi in sessantesimi di secondo. Per mantenere la durata massima di un evento al di sotto dei quattro secondi e venticinque e per non andare al di sotto di un sessantesimo, sono stati imposti dei limiti al metronomo.

```
PROGRAM:  14.RAINBOW 2

 10 PRINT"[CLEAR]":PRINT"[WHITE]";
    CHR$(142):POKE 53280,2
    :POKE 53281,11
 20 FOR I=54272 TO 54296:POKE I,0:NEXT
 30 POKE 54277,35:POKE 54278,199
 40 POKE 54296,15
 50 ME=1
```

```
60 PRINT " (UNA MELODIA DI [RVS]DATA
   [RVOFF])";"  METRONOMO=";INT(90/ME)
70 READ HF,LF,DU
80 IF DU=-1 GOTO 140
90 DU=DU*ME
100 POKE 54272,LF:POKE 54273,HF
   :POKE 54276,33:POKE 162,0
110 IF PEEK(162)<DU GOTO 110
120 POKE 54276,32
130 GOTO 70
140 PRINT:PRINT " PREMI <F1> PER $ENT
   IRLA PIU' VELOCE"
150 PRINT " O <F3> PER SENTIRLA PIU'
   LENTA"
160 PRINT " SE NO PREMI QUALSIASI ALT
   RO TASTO"
170 GET A$:IF A$="" GOTO 170
180 IF A$=CHR$(134) THEN ME=ME+0.1
   :IF ME>1.5 THEN ME=1.5:GOTO 200
190 IF A$=CHR$(133) THEN ME=ME-0.1
   :IF ME<0.1 THEN ME=0.1
200 PRINT"[CLEAR]":RESTORE:GOTO 60
210 DATA 17,37,80,34,75,80,32,94,20
220 DATA 25,177,40,28,214,20,32,94,40
230 DATA 34,75,40,17,37,80,28,214,80
240 DATA 25,177,160,-1,-1,-1
```

Questo programma non è perfettamente efficiente, e se lo mettete alla prova con un vero metronomo ve ne renderete conto. Supponiamo, infatti, che il computer abbia appena terminato di suonare la prima nota, avendo spento il gate nella linea 120. In teoria, dovrebbe subito far suonare la nota successiva. Invece deve fare tutta una serie di cose: 1) tornare alla linea 70 (e quando un programma Basic trova una GOTO, va a cercare la linea ripartendo da zero: quanto più è alto il numero di linea, tanto più ci mette a trovarla; 2) deve leggere i DATA; 3) deve controllare che il pezzo non sia finito (linea 80); 4) deve aggiornare la durata secondo il valore del metronomo; 5) deve aggiornare i valori della frequenza. Tutte queste operazioni, in Basic, portano via tempo; se ne può tenere conto, togliendo magari un sessantesimo dai valori delle durate memorizzati nei DATA (ad esempio, modificando la linea 90 CON DU=(DU*ME)−1): ma è chiaro che la vera soluzione consiste nel far eseguire tutte queste operazioni (o meglio, quelle equivalenti) da una routine in linguaggio macchina.

Quella che segue è una routine di questo tipo, e sta alla base del successivo programma *Sequencer*:

```
add.    data      source code
0                 10 PRT
0                 20 SYM
0                 30 LEGGE=49299
0                 40 SCRIVE=49300
```

```
0                    50 L1=54272
0                    60 L2=54279
0                    70 L3=54286
0                    80 H1=54273
0                    90 H2=54280
0                   100 H3=54287
0                   110 CR1=54276
0                   120 CR2=54283
0                   130 CR3=54290
0                   140 ORG$C000
C000  A200          150 INIT LDX#$00
C002  A900          160 NEWNOT LDA#$00
C004  85A1          170 STA$A1
C006  85A2          180 STA$A2
C008  BD00C1        190 LDA$C100.X
C00B  8D00D4        200 STA L1
C00E  BD00C2        210 LDA$C200.X
C011  8D07D4        220 STA L2
C014  BD00C3        230 LDA$C300.X
C017  8D0ED4        240 STA L3
C01A  BD00C4        250 LDA$C400.X
C01D  8D01D4        260 STA H1
C020  BD00C5        270 LDA$C500.X
C023  8D08D4        280 STA H2
C026  BD00C6        290 LDA$C600.X
C029  8D0FD4        300 STA H3
C02C  BD00C7        310 LDA$C700.X
C02F  8D04D4        320 STA CR1
C032  BD00C8        330 LDA$C800.X
C035  8D0BD4        340 STA CR2
C038  BD00C9        350 LDA$C900.X
C03B  8D12D4        360 STA CR3
C03E  BD00CA        370 LDA$CA00.X
C041  C5A1          380 CRON1 CMP$A1
C043  D0FC          390 BNE CRON1
C045  BD00CB        400 LDA$CB00.X
C048  C5A2          410 CRON2 CMP$A2
C04A  D0FC          420 BNE CRON2
C04C  BD00C7        430 LDA$C700.X
C04F  29FE          440 AND#$FE
C051  8D04D4        450 STA CR1
C054  BD00C8        460 LDA$C800.X
C057  29FE          470 AND#$FE
C059  8D0BD4        480 STA CR2
C05C  BD00C9        490 LDA$C900.X
C05F  29FE          500 AND#$FE
C061  8D12D4        510 STA CR3
C064  E4FB          520 CPX$FB
C066  F004          530 BEQ FINTES
C068  E8            540 INX
C069  4C02C0        550 JMP NEWNOT
C06C  20E4FF        560 FINTES JSR$FFE4
C06F  C900          570 CMP#$00
C071  F001          580 BEQ NEWSEQ
C073  60            590 RTS
C074  4C00C0        600 NEWSEQ JMP INIT
C077  78            610 SEI
C078  A984          620 LDA#$84
```

```
C07A  8D1403    630 STA 788
C07D  A9C0      640 LDA#$C0
C07F  8D1503    650 STA 789
C082  58        660 CLI
C083  60        670 RTS
C084  AC93C0    680 LDY LEGGE
C087  AE94C0    690 LDX SCRIVE
C08A  B900D4    700 LDA 54272.Y
C08D  9D00D4    710 STA 54272.X
C090  4C31EA    720 JMP$EA31
```

Per consentire di programmare durate anche superiori ai quattro secondi, la routine si serve di due bytes per controllare la durata, confrontandoli con i valori del cronometro nelle locazioni 161 e 162. Chi conosce il linguaggio Assembly noterà che la successione di eventi diversi è controllata con un impiego estensivo dell'indirizzamento assoluto indicizzato: tutti i dati (frequenze, durate, forme d'onda) sono memorizzati in sequenza in blocchi omogenei di memoria, e sono puntati attraverso il registro X. La JSR che compare verrà riconosciuta da chi traffica con il sistema operativo del Commodore 64 come un salto alla subroutine Kernal che legge un carattere dalla tastiera.

Il programma *Sequencer* che segue, pur nelle sue ragguardevoli dimensioni, non è altro che un "compilatore" di dati per questa routine. Permette di creare sequenze ripetitive di suoni e pause, definendo gli eventi passo su passo dalla tastiera del computer. Le sequenze sono monofoniche, ma è possibile accordare i tre oscillatori su frequenze diverse, per ottenere timbri complessi. Si tratta di un programma molto lungo: abbiate pazienza quando lo caricate. Le istruzioni per l'uso si trovano nell'Appendice.

Il Commodore 64, naturalmente, può lavorare anche in polifonia: lo rende possibile il fatto che ognuno dei tre oscillatori sia controllato da un segnale di gate indipendente. Non è difficile, quindi, "scrivere" composizioni a tre voci, memorizzando durate diverse per gli eventi rispettivi (il modo più razionale per farlo è di indicare con una sola successione di dati l'intervallo tra un evento e quello seguente, in qualsiasi voce si trovino, e di indicare con un altro dato in quale voce si verifica il cambiamento). Ma mentre un sintetizzatore monofonico a tre oscillatori è uno strumento dalle ottime risorse musicali, un sintetizzatore polifonico a tre voci è poco più di un giocattolo. Questa non vuole essere una giustificazione al fatto che, in conclusione di questo manuale, non venga presentato quello che alcuni ritengono sia il punto di arrivo delle capacità musicali del Commodore 64: un sequencer (o *editor musicale*) polifonico. La giustificazione più ovvia, come sempre, è lo spazio. Chi vi ha accompagnato fin qui si augura che ce ne sia ancora altro, presto, per andare avanti, e vi ringrazia della pazienza e dell'attenzione. (E non dimenticatevi di sentire il Finale!).

APPENDICE 1

FREQUENZE DELLE NOTE DELLA SCALA TEMPERATA, E LORO RAPPRESENTAZIONE

La tabella che segue presenta i valori delle frequenze delle note della scala temperata, all'interno dell'estensione del SID, con la loro rappresentazione in due bytes (alto e basso). La tabella differisce da quella fornita dalla Commodore, basata su un valore di clock di 1.023 MHz. Purtroppo le note della tabella Commodore sono calanti di circa un semitono, rendendo impossibile usare il SID assieme ad altri strumenti: dal che si deve concludere che o non sono esatti i fattori di conversione pubblicati sulla *Guida di riferimento*, oppure il valore del clock che si deve usare per i calcoli non è quello reale ma corrisponde effettivamente a 1 MHz (questa è la soluzione più probabile). Comunque sia, con i numeri della tabella pubblicata qui (arrotondati per il byte basso al valore più vicino) si ottiene un'intonazione perfetta con qualsiasi strumento ben accordato: e nessun informatico convincerà mai un musicista che un La è uguale a un Sol diesis.

OTTAVA 0				OTTAVA 1			
DO	16.351	1	18	DO	32.703	2	37
DO#	17.323	1	35	DO#	34.647	2	69
RE	18.354	1	52	RE	36.708	2	104
RE#	19.445	1	70	RE#	38.890	2	140
MI	20.601	1	90	MI	41.203	2	179
FA	21.826	1	110	FA	43.653	2	220
FA#	23.124	1	132	FA#	46.249	3	8
SOL	24.499	1	155	SOL	48.999	3	54
SOL#	25.956	1	179	SOL#	51.913	3	103
LA	27.499	1	205	LA	54.999	3	155
LA#	29.135	1	233	LA#	58.270	3	210
SI	30.867	2	6	SI	61.735	4	12

OTTAVA 2				OTTAVA 3			
DO	65.406	4	73	DO	130.812	8	147
DO#	69.295	4	139	DO#	138.591	9	21
RE	73.416	4	208	RE	146.832	9	159
RE#	77.781	5	25	RE#	155.563	10	50
MI	82.406	5	103	MI	164.813	10	205
FA	87.307	5	185	FA	174.614	11	114
FA#	92.498	6	16	FA#	184.997	12	32
SOL	97.998	6	108	SOL	195.997	12	216
SOL#	103.826	6	206	SOL#	207.652	13	156
LA	110.000	7	53	LA	220.000	14	107
LA#	116.540	7	163	LA#	233.081	15	70
SI	123.470	8	23	SI	246.941	16	47

OTTAVA 4

DO	261.625	17	37
DO#	277.182	18	42
RE	293.664	19	63
RE#	311.126	20	100
MI	329.627	21	154
FA	349.228	22	227
FA#	369.994	24	63
SOL	391.995	25	177
SOL#	415.304	27	56
LA	440.000	28	214
LA#	466.163	30	141
SI	493.883	32	94

OTTAVA 5

DO	523.251	34	75
DO#	554.365	36	85
RE	587.329	38	126
RE#	622.253	40	200
MI	659.255	43	52
FA	698.456	45	198
FA#	739.988	48	127
SOL	783.990	51	97
SOL#	830.609	54	111
LA	880.000	57	172
LA#	932.327	61	26
SI	987.766	64	188

OTTAVA 6

DO	1046.502	68	149
DO#	1108.730	72	169
RE	1174.659	76	252
RE#	1244.507	81	143
MI	1318.510	86	105
FA	1396.912	91	140
FA#	1479.977	96	254
SOL	1567.981	102	194
SOL#	1661.218	108	223
LA	1760.000	115	88
LA#	1864.655	122	52
SI	1975.533	129	120

OTTAVA 7

DO	2093.004	137	43
DO#	2217.461	145	83
RE	2349.318	153	247
RE#	2489.015	163	31
MI	2637.020	172	210
FA	2793.825	183	25
FA#	2959.955	193	252
SOL	3135.963	205	133
SOL#	3322.437	217	189
LA	3520.000	230	176
LA#	3729.310	244	103

APPENDICE 2

ISTRUZIONI PER L'USO DEL SEQUENCER

Un sequencer permette di memorizzare sequenze di suoni e pause in una forma "leggibile" da un sintetizzatore. L'uso più tipico dei sequencer nella musica pop consiste nella creazione di sequenze ripetitive, che possono essere usate come *riff*. Se la memoria del sequencer è abbastanza ampia, però, si possono creare interi brani musicali; in questo caso il sequencer, specialmente se polifonico, diventa un vero strumento di composizione.

Si distinguono due tipi di sequencer: sequencer in tempo reale, che memorizzano quanto viene suonato sulla tastiera e sono in grado poi di riprodurre l'esecuzione, e *step-sequencer*, con i quali le note vengono introdotte una dopo l'altra, specificando l'altezza, la durata e altri parametri.

Quello che trovate sulla cassetta è uno step-sequencer, con una capacità di 256 eventi (un evento è una nota o una pausa). Si tratta di un sequencer monofonico a tre oscillatori, che permette di usare tutte le caratteristiche di generazione del suono del SID (compresa la modulazione ad anello e i controlli dell'oscillatore 3, assegnabili a qualsiasi parametro). Viene controllato da due menu principali, uno che riguarda la "scrittura" della sequenza, l'altro che riguarda le caratteristiche dell'esecuzione. Il sequencer funziona in modo ciclico, ma è possibile anche ascoltare la sequenza una sola volta.

Il programma assiste l'utilizzatore in due modi: 1) spiegando nella parte bassa dello schermo il significato delle domande e le possibili risposte; 2) fornendo nella maggior parte dei casi delle risposte già pronte, riducendo così la possibilità di errori di battitura e facendo risparmiare tempo. Quello che non trovate in queste istruzioni, quindi, lo troverete probabilmente nel programma stesso.

Quando si dà il RUN, dopo pochi secondi, appare il menu che controlla le funzioni di scrittura.

Queste sono:

1. SCRITTURA
2. CARICAMENTO
3. MODIFICA
4. CANCELLAZIONE
5. MEMORIZZAZIONE
6. ESECUZIONE
7. LETTURA O STAMPA
8. FINE PROGRAMMA
9. INIT DISK

Quando usate il programma per la prima volta potete scegliere solo

SCRITTURA, FINE PROGRAMMA, INIT DISK. Non avete infatti sequenze da caricare (da cassetta o disco), e non ci sono sequenze in memoria. Se provate a scegliere altre funzioni, il programma vi riporterà al menù dopo avervi avvertito che non ci sono sequenze in memoria.

SCRITTURA è, ovviamente, la fase più importante di tutte. Permette di fornire i dati della sequenza che dovrà essere eseguita. Appena si sceglie questa opzione, compare la domanda PUNTO D'INIZIO. Questo è il punto da cui si inizia a scrivere: quando si lancia il programma e si entra subito in SCRITTURA il punto d'inizio è naturalmente 1 (il primo evento), ma se si esce dalla SCRITTURA e poi si rientra, o se in memoria c'è già una sequenza caricata da cassetta o disco, allora il punto di inizio è il primo evento dopo l'ultimo della sequenza già in memoria. In questo modo si può proseguire un lavoro interrotto o sommare una nuova sequenza a un'altra. L'unica condizione è che la lunghezza totale non superi i 256 eventi. Si può anche spostare il punto d'inizio all'indietro, e cancellare una parte della sequenza già in memoria.

Scelto il punto d'inizio compare l'indicazione dell'evento che si va ad inserire: sarà una nota o una pausa (a questo punto si può anche uscire dalla SCRITTURA, rispondendo F (per Fine)).

Se la risposta è N (per Nota), segue la richiesta SCALA TEMPERATA? Se si risponde negativamente il programma richiede la frequenza esatta in Hertz che si vuole assegnare alla nota; altrimenti alla domanda successiva (QUALE NOTA?) si dovrà rispondere indicando il nome della nota e, ancora dopo, l'ottava in cui si trova (ricordando che il Do centrale è la prima nota della quarta ottava).

Segue la richiesta della forma d'onda (da scegliere tra le quattro principali del SID), e poi della durata. La durata deve essere indicata in centoventottesimi: un quarto è 32, un ottavo 16, un intero 128. Il programma accetta anche valori frazionari, permettendo di realizzare figurazioni irregolari: ad esempio ogni nota di una terzina di ottavi (tre ottavi nella durata di un quarto) si può indicare con 10.666 (32/3).

Il programma suggerisce sempre dei valori a tutte queste risposte. Alla scelta tra nota o pausa suggerisce sempre nota, e all'indicazione della frequenza suggerisce la scala temperata, mentre per gli altri valori ripropone quelli che voi avete inserito la volta precedente: in questo modo, se siete in una certa ottava, non dovrete battere il valore dell'ottava a ogni nota diversa, ma vi basterà premere (RETURN); allo stesso modo, per scrivere una successione di terzine non dovrete ogni volta ribattere il valore. Per fare una lunga serie di note ribattute non bisogna far altro che tenere premuto il tasto (RETURN).

Attenzione, però: il Commodore 64 accetta in input tutti i caratteri che si trovano sulla riga dopo il cursore, per cui, se una nota prima avete scritto SOL e volete introdurre ora un LA, dovete battere uno spazio per cancellare la L di SOL; altrimenti il computer leggerà la "nota" LAL e vi riproporrà la domanda.

Una volta scritta la sequenza, e tornati al menù principale, si può leg-

gere quello che si è scritto scegliendo l'opzione LETTURA o STAMPA. Si può vedere la sequenza sul video, in cinque colonne: da sinistra il numero dell'evento, il nome della nota (uno spazio vuoto se è una pausa, i caratteri * * * se non è una frequenza temperata), la frequenza (uno spazio vuoto se è una pausa), la durata in 128-esimi, la forma d'onda (espressa col numero del registro di controllo: 33 per l'onda a dente di sega, 17 per la triangolare eccetera). Se la sequenza è troppo lunga per stare sul video il computer si ferma e aspetta che premiate un tasto qualsiasi per farvi vedere un altro schermo; lo stesso avviene alla fine, prima di tornare al menu.

Con la stampante dovrete fare le stesse operazioni.

Se la sequenza non vi soddisfa potete scegliere la MODIFICA. Si possono scegliere due modi: l'inserimento o la sostituzione. Nel primo caso potete cancellare un evento e sostituirlo con un altro (l'operazione avviene nello stesso momento), nel secondo potete inserire un evento tra altri due. Nel primo caso il programma chiede quale evento si vuol modificare, nel secondo chiede prima di quale ci si vuole inserire. Anche all'interno di questa opzione è possibile leggere la sequenza. Un altro modo di alterare la frequenza è quello della CANCELLAZIONE. Oltre a cancellare tutto si può anche togliere dalla sequenza un singolo evento.

Se la sequenza è a posto, si può passare all'ESECUZIONE. Ci sono due modi di esecuzione; uno è il modo standard, che permette di riascoltare entro un tempo ragionevolmente breve la sequenza, ma senza specificare i parametri timbrici né il metronomo, che rimane fisso a 60 (quarti al minuto). L'altro è il modo più complesso, che avviene dopo aver selezionato tutti i parametri da un nuovo menu.

Queste sono le scelte:

1. INVILUPPO
2. MODULAZIONE ONDA QUADRA
3. NUMERO E ACCORDATURA OSCILLATORI
4. FILTRO E RISONANZA
5. MODULAZIONE AD ANELLO E SYNC
6. CONTROLLO DA OSC 3 E ENV 3
7. METRONOMO
8. ESECUZIONE SEQUENZA
9. RITORNO AL MENU PRINCIPALE

Ognuna di queste opzioni permette di scegliere i parametri che si preferiscono per l'esecuzione. Il programma accetta inviluppi distinti per i tre oscillatori e tutte le percentuali di modulazione dell'onda quadra possibili (anche diverse per i tre oscillatori). L'opzione 3 permette di definire la frequenza effettiva degli oscillatori, che può anche essere diversa da quella fissata nella sequenza: vale a dire che la frequenza di ogni evento può essere moltiplicata per un fattore (uguale per tutta la sequenza, ma diverso da oscillatore a oscillatore). In questo modo si può, ad esempio, trasportare tutta una sequenza in su o in giù (un rapporto

di frequenza 2 porta tutto un'ottava sopra, un rapporto 1.5 porta tutto una quinta sopra); si possono ottenere accordi (ponendo un oscillatore a 1, uno a 1.25 e uno a 1.5 si ottiene un accordo maggiore: vedete il programma SCALE per i rapporti degli altri intervalli); si possono ottenere effetti di *detune* e di *phasing* (provate, ad esempio, i valori 1/1.01/0.99).

Le altre opzioni non dovrebbero creare problemi, se avete letto il libro; il CONTROLLO DA OSC 3 E ENV 3 funziona nel modo descritto nel penultimo capitolo, e prevede che si fornisca il numero del registro del SID da controllare. Nel caso che si impieghi l'oscillatore 3 come LFO viene richiesta la forma d'onda. I valori di *default* (quelli forniti dal programma in partenza) prevedono il controllo sul filtro: ricordate, però, che per sentire l'effetto è necessario attivare il filtro stesso (opzione 4).

Gli effetti più curiosi si ottengono con la modulazione ad anello; ricordate però che per ottenerli dovete accordare gli oscillatori in modo diverso (opzione 3): infatti, la modulazione ad anello di due note uguali è... ancora la stessa nota (un'ottava sopra).

Il metronomo si definisce nel modo classico: in unità di tempo (in questo caso quarti) al secondo. Non ci sono valori massimi, anche se un metronomo troppo veloce può far perdere alla sequenza la sua fisionomia ritmica: al di sotto del sessantesimo di secondo il programma uguaglia tutte le durate. Tenete ben conto che con metronomi veloci e tempi di attacco o decadimento lunghi potrebbe diventare difficile distinguere la fine di una nota dall'inizio di quella successiva . Il metronomo è protetto da valori troppo bassi; per questo (come nel caso di altre opzioni, che vedrete) il computer può prendere un po' di tempo prima di accettare il valore proposto, specialmente se la sequenza è lunga.

Nel modo di esecuzione di questo menu, come in quello standard, il computer prima di partire deve compilare tutti i dati che gli avete fornito; questo può richiedere un certo tempo. La sequenza viene fatta partire premendo un tasto, e viene interrotta (ma solo alla fine, notate bene) premendo ancora un tasto. Se si vuole risentire ancora la sequenza, questa volta si può farlo subito, perché ormai la compilazione è stata eseguita.

Tornando al menu principale, ora è possibile memorizzare la sequenza. Se avete solo il registratore a cassette non dovete far altro che rispondere alle domande e inserire una cassetta (possibilmente, per vostra comodità, una cassetta breve per ogni sequenza); se avete un disk drive scegliete l'opzione INIT DISK, che crea sul disco un file nel quale saranno memorizzati i nomi di tutte le sequenze. Potete usare a questo scopo lo stesso disco sul quale conserverete il programma, ricordando però che, per comodità di lettura, su un disco stanno non più di 40 sequenze. Tenete presente che se non eseguite questa parte del programma e cercate di memorizzare la sequenza otterrete un messaggio di errore e la perdita della sequenza in memoria. Viceversa, se inizializzate in questo modo un disco che conteneva già delle sequenze le perderete tutte (a meno che non vi ricostruiate un file "SEQUENZE", contenente al

primo posto il numero delle sequenze e quindi tutti i nomi in fila).

La MEMORIZZAZIONE registra sul supporto che avete scelto tutta la sequenza e tutti i parametri esecutivi (dell'ultima volta in cui l'avete ascoltata in modo non-standard). In questo modo potete conservarvi tutti i vostri *presets* timbrici.

Il CARICAMENTO vi permetterà poi di riintrodurre in memoria le sequenze desiderate; nel caso usiate un disk drive i nomi delle sequenze memorizzate appariranno sullo schermo per facilitarvi nella scelta. Se cercate di caricare una sequenza con un nome sbagliato il programma vi protegge dall'errore; allo stesso modo, non riuscirete a memorizzare su disco due sequenze con lo stesso nome. Alternando fasi di SCRITTURA, MEMORIZZAZIONE e CARICAMENTO potete scrivere in breve tempo sequenze molto lunghe, inserendo uno dopo l'altro blocchi diversi. Ricordate che i parametri timbrici dell'esecuzione finale saranno quelli dell'ultima sequenza caricata in memoria. Un trucco per conservare una serie di presets può essere questo: 1) definire i parametri su una sequenza lunga; 2) memorizzare la sequenza; 3) cancellarla tutta (la CANCELLAZIONE toglie dalla memoria la sequenza, ma non i parametri esecutivi); 4) scrivere una sequenza di una sola nota; 5) memorizzarla. Tutte le volte che vorrete richiamare questi parametri non dovrete fare altro che caricare quest'ultima sequenza in coda a quella che avete in memoria, cancellare l'ultimo evento, suonare.

Buon ascolto!

TABELLA DEI REGISTRI DEL SID

		BIT n.	7	6	5	4	3	2	1	0
		DECIM.	128	64	32	16	8	4	2	1
INDIRIZZO		FUNZIONE	**OSCILLATORE 1**							
0	54272	LO FQ	f 7	f 6	f 5	f 4	f 3	f 2	f 1	f 0
1	54273	HI FQ	f 15	f 14	f 13	f 12	f 11	f 10	f 9	f 8
2	54274	LO PW (PL)	pw 7	pw 6	pw 5	pw 4	pw 3	pw 2	pw 1	pw 0
3	54275	HI PW (PH)	/////	/////	/////	/////	pw 11	pw 10	pw 9	pw 8
4	54276	REG. CONT.	RUM.				TEST	R MOD	SYNC	GATE
5	54277	ATT–DEC	ATT 3	ATT 2	ATT 1	ATT 0	DEC 3	DEC 2	DEC 1	DEC 0
6	54278	SOS–RIL	SOS 3	SOS 2	SOS 1	SOS 0	RIL 3	RIL 2	RIL 1	RIL 0
			OSCILLATORE 2							
7	54279	LO FQ	f 7	f 6	f 5	f 4	f 3	f 2	f 1	f 1
8	54280	HI FQ	f 15	f 14	f 13	f 12	f 11	f 10	f 9	f 8
9	54281	LO PW (PL)	pw 7	pw 6	pw 5	pw 4	pw 3	pw 2	pw 1	pw 0
10	54282	HI PW (PH)	/////	/////	/////	/////	pw 11	pw 10	pw 9	pw 8
11	54283	REG. CONT.	RUM.				TEST	R MOD	SYNC	GATE
12	54284	ATT–DEC	ATT 3	ATT 2	ATT 1	ATT 0	DEC 3	DEC 2	DEC 1	DEC 0
13	54285	SOS–RIL	SOS 3	SOS 2	SOS 1	SOS 0	RIL 3	RIL 2	RIL 1	RIL 0
			OSCILLATORE 3							
14	54286	LO FQ	f 7	f 6	f 5	f 4	f 3	f 2	f 1	f 0
15	54287	HI FQ	f 15	f 14	f 13	f 12	f 11	f 10	f 9	f 8
16	54288	LO PW (PL)	pw 7	pw 6	pw 5	pw 4	pw 3	pw 2	pw 1	pw 0
17	54289	HI PW (PH)	/////	/////	/////	/////	pw 11	pw 10	pw 9	pw 8
18	54290	REG. CONT.	RUM.				TEST	R MOD	SYNC	GATE
19	54291	ATT–DEC	ATT 3	ATT 2	ATT 1	ATT 0	DEC 3	DEC 2	DEC 1	DEC 0
20	54292	SOS–RIL	SOS 3	SOS 2	SOS 1	SOS 0	RIL 3	RIL 2	RIL 1	RIL 0
			FILTRO, VOLUME E REGISTRI DI LETTURA							
21	54293	LO TN (TL)	/////	/////	/////	/////	/////	tn 2	tn 1	tn 0
22	54294	HI TN (TH)	tn 10	tn 9	tn 8	tn 7	tn 6	tn 5	tn 4	tn 3
23	54295	RIS–FILT	RIS 3	RIS 2	RIS 1	RIS 0	FILTEX	FILT 3	FILT 2	FILT 1
24	54296	VOL–MODO	3 OFF	HP	BP	LP	VOL 3	VOL 2	VOL 1	VOL 0
25	54297	POT 1	7	6	5	4	3	2	1	0
26	54298	POT 2	7	6	5	4	3	2	1	0
27	54299	OSC 3	O 7	O 6	O 5	O 4	O 3	O 2	O 1	O 0
28	54300	ENV 3	E 7	E 6	E 5	E 4	E 3	E 2	E 1	E 0

Beatrice d'Este
ARTI GRAFICHE RICORDI